

PROBLEMA DE CLASIFICACIÓN MULTI-CLASE: RNA-seq

Martina Álvarez Lorenzo

Contents

1	Introducción	4
2	Preparación de los datos	6
2.1	Comparativa para distintos valores de CLF y COV	10
3	Comparativas algoritmo de selección de características y clasificación	18
3.1	Algoritmo de clasificación: k-NN	19
3.1.1	Algoritmo de selección de características: mRMR	19
3.1.2	Algoritmo de selección de características: RF	24
3.1.3	Algoritmo de selección de características: DA	29
3.1.4	Algoritmo de selección de características: Lasso	34
3.1.5	Conclusiones	39
3.2	Algoritmo de clasificación: Random Forest	39
3.2.1	Algoritmo de selección de características: mRMR	39
3.2.2	Algoritmo de selección de características: RF	43
3.2.3	Algoritmo de selección de características: DA	47
3.2.4	Algoritmo de selección de características: Lasso	51
3.2.5	Conclusiones	56
3.3	Algoritmo de clasificación: SVM	56
3.3.1	Algoritmo de selección de características: mRMR	56
3.3.2	Algoritmo de selección de características: RF	60
3.3.3	Algoritmo de selección de características: DA	64
3.3.4	Algoritmo de selección de características: Lasso	68
3.3.5	Conclusiones	72
3.4	Algoritmo de clasificación: Redes neuronales	72
3.4.1	Algoritmo de selección de características: mRMR	73
3.4.2	Algoritmo de selección de características: RF	85
3.4.3	Algoritmo de selección de características: DA	89
3.4.4	Algoritmo de selección de características: Lasso	93
3.4.5	Conclusiones	97
3.5	Elecciones finales	97
4	Evolución del rendimiento según el número de genes utilizando 5-CV con el clasificador seleccionado	100
4.1	Resultado de la huella final y clasificador en 5CV	110
5	Resultado de la huella final escogida	117
6	Comparación con huella externa	122
6.1	Conclusiones	126

7	Enriquecimiento de genes para la huella seleccionada	127
7.1	Gene Ontology (GO)	127
7.2	Pathways	147
7.3	Disease Association	149
7.3.1	LINP1	149
7.3.2	LHX6	150
7.3.3	APLN	152

1 Introducción

El cáncer de hígado es una enfermedad grave que afecta a muchas personas en todo el mundo, con una incidencia y mortalidad significativamente más altas en ciertas regiones. Según datos de la Sociedad Americana Contra el Cáncer, se estima que en 2024 se reportarán aproximadamente 41.630 nuevos casos de cáncer primario de hígado y cáncer de las vías biliares intrahepáticas en los Estados Unidos, con alrededor de 29.840 muertes asociadas (<https://www.cancer.org/es/cancer/tipos/cancer-de-higado/acerca/que-es-estadisticas-clave.html>).

A nivel global, en 2020, se diagnosticaron aproximadamente 905.700 casos nuevos y se registraron 830.200 muertes por cáncer de hígado. Las tasas de incidencia y mortalidad fueron más altas en regiones como Asia oriental, África del Norte y el sudeste asiático (<https://asscat-hepatitis.org/incidencia-mundial-de-cancer-primario-de-higado-en-2020-y-prevision-global-para-2040/>).

Uno de los métodos para estudiar el cáncer es analizar los genes, ya que estos proporcionan información clave sobre cómo se desarrolla la enfermedad. Cada célula de nuestro cuerpo tiene un conjunto de genes que controlan su funcionamiento, y cuando estos genes se alteran, pueden provocar enfermedades como el cáncer.

En este proyecto, nos enfocaremos en el cáncer de hígado, específicamente en tres grupos de muestras de tejido: **Solid Tissue Normal**, **TCGA-CHOL** y **TCGA-LIHC**. Estas tres clases representan diferentes tipos de muestras de tejido: *tejido de hígado sano*, *tumores de colangiocarcinoma* (un tipo de cáncer de las vías biliares) y *tumores de carcinoma hepatocelular* (el tipo más común de cáncer de hígado).

La estadificación del cáncer de hígado es importante para determinar el pronóstico y las opciones de tratamiento. El sistema de estadificación BCLC (Barcelona Clinic Liver Cancer) clasifica el cáncer de hígado en cinco estadios:

- **Estadio 0 (Muy Temprano):** Un solo tumor pequeño sin invasión vascular.
- **Estadio A (Temprano):** Un solo tumor o múltiples tumores pequeños sin invasión vascular.
- **Estadio B (Intermedio):** Múltiples tumores de mayor tamaño sin invasión vascular.
- **Estadio C (Avanzado):** Invasión vascular o metástasis a órganos distantes.
- **Estadio D (Terminal):** Enfermedad muy avanzada con afectación generalizada.

Comprender la incidencia y las características de los diferentes tipos de cáncer de hígado, así como los estadios de la enfermedad, es importante para desarrollar modelos predictivos efectivos y mejorar los resultados en el diagnóstico y tratamiento del cáncer de hígado.

En el siguiente apartado se detallará el proceso de preparación de los datos y se presentará la cantidad de muestras correspondientes a cada una de las clases que se analizarán.

El objetivo principal de este proyecto es desarrollar un modelo capaz de clasificar correctamente las muestras en tres clases: **tejido normal sano** (Solid Tissue Normal), **tumores de colangiocarcinoma** (TCGA-CHOL) y **tumores de carcinoma hepatocelular** (TCGA-LIHC). Para ello, emplearemos varios algoritmos de clasificación, incluyendo **K-Nearest Neighbors** (KNN), **Random Forest** (RF), **Support Vector Machine** (SVM) y **Redes Neuronales** (RRNN).

Además, para mejorar la precisión del modelo y seleccionar las características más relevantes, utilizaremos diversos algoritmos de selección de características: *mRMR*, *RF*, *DA* y *Lasso*. Estos métodos nos permitirán identificar los genes más importantes para diferenciar entre los tres tipos de tejido, mejorando así la capacidad predictiva de nuestros modelos.

Además de los algoritmos de selección de características y clasificación que proporciona KnowSeq, se ha decidido implementar los siguientes con el objetivo de obtener una mayor robustez, específicamente:

- **Lasso:** Es un método de regresión que combina la minimización del error con una penalización que elimina características menos relevantes, logrando una selección automática de variables. Este enfoque reduce el ruido en los datos al descartar variables redundantes, manteniendo únicamente las más significativas. En este proyecto, Lasso se utiliza para identificar los genes más influyentes en la clasificación de las muestras. Su capacidad para manejar conjuntos de datos con alta dimensionalidad (aunque en este caso el número de muestras no sea muy grande) lo hace particularmente adecuado para análisis genéticos, garantizando un modelo más eficiente y con mayor capacidad predictiva.
- **Redes Neuronales:** Son modelos avanzados de aprendizaje automático que simulan el funcionamiento del cerebro humano. Están compuestas por capas de neuronas conectadas, cuyos pesos se ajustan mediante métodos de optimización como el gradiente descendente. Durante el entrenamiento, las RRNN identifican patrones complejos, incluso no lineales, en los datos, lo que permite una clasificación precisa. Este modelo es especialmente útil para el análisis de datos genéticos relacionados con el cáncer de hígado, ya que puede procesar grandes volúmenes de información (aunque en este caso el número de muestras no sea muy grande) y descubrir patrones ocultos que diferencien las tres clases de tejido. Además, las RRNN son flexibles y escalables, lo que las hace ideales para abordar problemas complejos como este.

Este análisis de este tipo es importante, ya que un modelo que pueda diferenciar entre estos tres tipos de tejido puede ayudar a los médicos a diagnosticar el cáncer de hígado con mayor rapidez, evaluar el estado de la enfermedad y determinar los tratamientos más adecuados para cada paciente.

2 Preparación de los datos

Los datos utilizados en esta práctica se han obtenido del *GDC Data Portal Homepage*. En esta plataforma, se seleccionó el cáncer de hígado como área de interés y, siguiendo las instrucciones proporcionadas por los tutores, se creó una cohorte con información específica para abordar el problema de clasificación en tres clases. Tras definir las características necesarias, se generó un repositorio siguiendo los pasos indicados en la asignatura. Como resultado, se deberían obtener 464 archivos.

Dado el volumen de datos, fue necesario descargar el archivo **Manifest** desde el portal y utilizar la herramienta `gdc-client` para gestionar y ejecutar la descarga de las muestras. Además, se descargó el archivo **Sample Sheet**, para identificar las características de las muestras. Una vez preparados todos los archivos, se procedió a configurar los datos para su análisis.

El primer paso en la configuración consiste en incluir el paquete `KnowSeq` y las funciones proporcionadas.

El siguiente paso consiste en identificar el directorio donde se encuentran los ficheros de los pacientes y convertir los datos de counts a un formato adecuado para el análisis.

```
down_folder = "liver_data/"
sample_sheet = "gdc_sample_sheet.2024-12-25.tsv"
samples <- convert_to_counts(down_folder, sample_sheet, outpath = "liver_data/counts",
                             column_name = 'unstranded', filter_lines = 4)
```

Posteriormente, se crearán las variables necesarias para construir un dataframe que contenga la información de los ficheros procesados. Estas variables serán:

- **Run:** almacenará la ruta completa de cada archivo.
- **Path:** contendrá la ubicación del directorio donde se encuentran los ficheros.
- **Class:** identificará la clase de cada muestra, asignándola a una de las tres categorías: *LIHC*, *CHOL* o *Healthy*.

```
Run <- samples$File.Name
Path <- rep("liver_data/counts", nrow(samples))
Class <- samples$Sample.Type
Class[samples$Project.ID == "TCGA-LIHC"] <- "LIHC"
Class[samples$Project.ID == "TCGA-CHOL"] <- "CHOL"
Class[samples$Sample.Type == "Solid Tissue Normal"] <- "Healthy"
```

En primer lugar, mostramos el número de muestras presentes en cada clase del conjunto de datos:

```
table(Class)
```

```
## Class
##      CHOL Healthy      LIHC
##       32       58      374
```

Como se puede observar, el conjunto de datos contiene un total de 464 muestras distribuidas en tres clases:

- **Healthy** (muestras sanas): **58 muestras**
- **LIHC** (tumores de carcinoma hepatocelular): **374 muestras**
- **CHOL** (tumores de colangiocarcinoma): **32 muestras**

Podemos decir que el conjunto de datos presenta un claro desbalanceo en las clases, ya que las muestras **no están distribuidas equitativamente entre las tres categorías**. La mayoría de las muestras pertenecen a la clase LIHC (374 muestras), mientras que las clases Healthy y CHOL están significativamente menos representadas, con 58 y 32 muestras respectivamente.

El desbalanceo de clases puede suponer un reto en los estudios de clasificación, ya que los modelos tienden a favorecer la clase mayoritaria, lo que puede conducir a predicciones menos precisas para las clases minoritarias. Sin embargo, en este caso, debido a los buenos resultados obtenidos en la clasificación (se verá en próximos apartados), **no ha sido necesario aplicar técnicas de corrección del desbalanceo** como la sobremuestreo de las clases minoritarias (por ejemplo, con SMOTE) o el submuestreo de la clase mayoritaria.

Esto indica que el modelo ha sido capaz de capturar patrones significativos en los datos, incluso con el desbalanceo presente, y ha logrado un buen rendimiento general en todas las clases. Aun así, es importante verificar que las métricas de evaluación reflejen un buen desempeño en las clases minoritarias, para confirmar que el modelo no las ha ignorado, esto último se comprueba en el apartado correspondiente.

Creamos un DataFrame que contiene las variables `Run`, `Path` y `Class` para exportarlo como un archivo CSV. Esto permite conservar y reutilizar esta información en futuras etapas del análisis:

```
data.info <- data.frame(Run = Run, Path = Path, Class = Class)
write.csv(file = "data_info.csv", x = data.info)
```

A continuación, se cargarán y unirán los ficheros `counts`, generando una matriz que será exportada para agilizar futuras ejecuciones sin necesidad de repetir este paso:

```
countsInfo <- countsToMatrix("data_info.csv", extension = "")
save(countsInfo,file="countsInfo")
```

Posteriormente, cargamos la matriz de conteos guardada:

```
load(file="countsInfo")
```

Extraemos la matriz de conteos y las etiquetas de las muestras a partir del objeto `countsInfo` creado previamente:

```
countsMatrix <- countsInfo$countsMatrix
labels <- countsInfo$labels
```

En este paso, consultamos los **Gene Symbols** y el contenido de **CG** de cada gen utilizando los nombres de las filas de la matriz de conteos:

```
myAnnotation <- getGenesAnnotation(rownames(countsMatrix))
```

Se calculan los valores de **expresión génica** utilizando la matriz de conteos (`countsMatrix`) y la anotación obtenida en el paso anterior. Durante este proceso se normalizan los datos y se eliminan outliers, lo que permite identificar los genes diferencialmente expresados:

```
geneExprMatrix <- calculateGeneExpressionValues(countsMatrix, annotation = myAnnotation)
```

A continuación, eliminamos las filas sin anotación de gen y guardamos el resultado en un archivo para agilizar las futuras ejecuciones:

```
geneExprMatrix <- geneExprMatrix[!is.na(rownames(geneExprMatrix)),]
save(geneExprMatrix,file="geneExprMatrix")
```

Cargamos la matriz de expresión guardada:

```
load(file="geneExprMatrix")
```

En el siguiente paso, se realiza un análisis de calidad para filtrar las muestras consideradas outliers. Las muestras que pasan el filtro se guardan en las variables `qualityMatrix` y `qualityLabels`:

```
QAResults <- RNAseqQA(geneExprMatrix, toRemoval = TRUE, toPNG=FALSE, toPDF=FALSE)
```

Se seleccionan las muestras que pasen el filtro, es decir que no son outliers y se guardarán en las variables `qualityMatrix` y `qualityLabels`.


```
qualityMatrix <- QAResults$matrix
qualityLabels <- labels[-which(colnames(geneExprMatrix) %in% QAResults$outliers)]
```

```
save(qualityMatrix,file="qualityMatrix")
save(qualityLabels,file="qualityLabels")
```

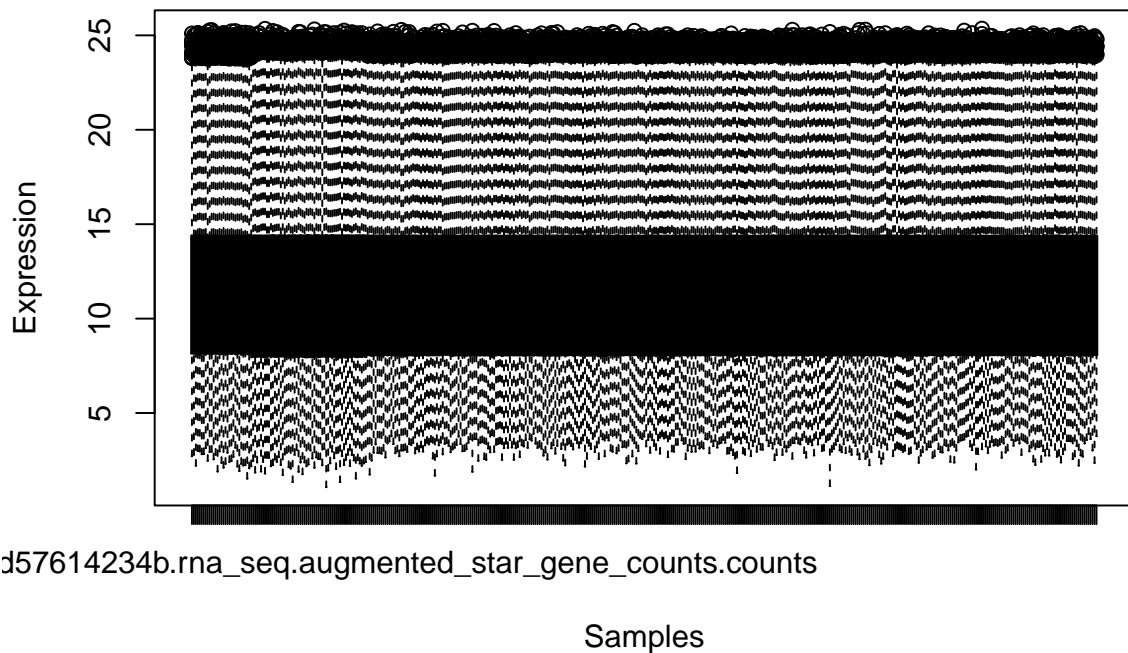
```
load(file="qualityMatrix")
load(file="qualityLabels")
```

Creamos un modelo SVA para corregir el efecto batch, obteniendo una matriz ajustada. Esta matriz se guarda para su uso en etapas posteriores:

```
batchMatrix <- batchEffectRemoval(qualityMatrix, qualityLabels, method = "sva")
save(batchMatrix,file="batchMatrix")
```

Una vez que se ha guardado, la cargaremos y dibujaremos el dataPlot para ver el resultado obtenido:

```
load(file="batchMatrix")
dataPlot(batchMatrix, qualityLabels, mode = "orderedBoxplot")
```



Una vez obtenido el resultado final, guardamos la matriz de expresión ajustada (`batchMatrix`) y las etiquetas de clase (`qualityLabels`) en nuevas variables:

```

MATRIZ <- batchMatrix
LABELS <- qualityLabels
save(MATRIZ, file = 'MATRIZ')
save(LABELS, file = 'LABELS')

```

Finalmente, cargamos la matriz de expresión final y las etiquetas para continuar con los análisis posteriores:

```

load(file = 'MATRIZ')
load(file = 'LABELS')
rownames(MATRIZ) <- make.names(rownames(MATRIZ))

```

2.1 Comparativa para distintos valores de CLF y COV

Este apartado tiene como objetivo preparar los datos para realizar la selección de un conjunto inicial de genes diferencialmente expresados (DEGs) y definir una metodología que permita evaluar los resultados de forma estructurada. Se hará lo siguiente:

1. Dividir los datos en conjunto de entrenamiento (Trn) y prueba (Test).
2. Identificar genes diferencialmente expresados (DEGs) usando LFC, COV y p-value.
3. Probar diferentes combinaciones de parámetros para obtener un conjunto inicial de genes razonable (50-300 genes).
4. Configurar la validación cruzada para evaluar modelos y seleccionar la huella génica final.

Se establecerá de antemano la semilla con el valor **7** para garantizar la reproducibilidad de los resultados.

```

set.seed(7)
nfolds <- 5
folds <- cvGenStratified(LABELS,nfolds)

nData <- dim(MATRIZ)[1]
indexTest <- which(folds==1)
indexTrn <- which(folds!=1)
XTrn <- MATRIZ[,indexTrn]
XTest <- MATRIZ[,indexTest]
YTrn <- LABELS[indexTrn]
YTest <- LABELS[indexTest]

```

Se van a definir varios valores de LFC y COV para ver que combinación da mejores resultados, ejecutando la función **DEGsExtraction** para identificar genes diferencialmente expresados (DEGs) en una matriz de datos. Para cada combinación, obtiene la cantidad de genes seleccionados (que será el número de filas de la matriz `DEGs_Matrix`) y guarda los resultados en una tabla con tres columnas: LFC, COV y NumGenes. Además, he incluido un try-catch para evitar interrupciones en caso de que alguna combinación falle (por si alguna no tiene genes), asignando 0 genes en esos casos y mostrando un mensaje informativo.

```
# Valores de LFC y COV para probar
lfc_values <- c(1.0, 1.25, 1.5, 1.75, 2.0)
cov_values <- c(1.0, 2.0)

# DataFrame para guardar resultados
results <- data.frame(LFC = numeric(0), COV = numeric(0), NumGenes = numeric(0))

# Bucle para probar combinaciones de LFC y COV
for (lfc in lfc_values) {
  for (cov in cov_values) {
    tryCatch({
      # Extraer genes diferencialmente expresados
      DEGsInfo <- DEGsExtraction(XTrn, YTrn, lfc = lfc, cov = cov, pvalue = 0.001)

      # Comprobar si la salida contiene la matriz esperada
      if (!is.null(DEGsInfo$DEG_Results) && "DEGs_Matrix" %in%
          names(DEGsInfo$DEG_Results)) {
        DEGsMatrix <- DEGsInfo$DEG_Results$DEGs_Matrix
        num_genes <- if (!is.null(DEGsMatrix)) dim(DEGsMatrix)[1] else 0
      } else {
        num_genes <- 0
      }

      # Guardar los resultados
      results <- rbind(results, data.frame(LFC = lfc, COV = cov, NumGenes = num_genes))

    }, error = function(e) {
      # Si hay error, asignar 0 genes y continuar
      results <- rbind(results, data.frame(LFC = lfc, COV = cov, NumGenes = 0))
      message(paste("Error en LFC =", lfc, "COV =", cov, ":", e$message))
    })
  }
}
```

```
}
```

```
## Error en LFC = 1.75 COV = 2 : los argumentos implican un número diferente de filas: 1, 0
```

```
## Error en LFC = 2 COV = 2 : There are not genes that complains these restrictions, please change t
```

A continuación se muestran los resultados obtenidos:

```
results
```

```
##      LFC COV NumGenes
## 1  1.00  1     1061
## 2  1.00  2        51
## 3  1.25  1     549
## 4  1.25  2         9
## 5  1.50  1     287
## 6  1.50  2         2
## 7  1.75  1     172
## 8  1.75  2         0
## 9  2.00  1     114
## 10 2.00  2         0
```

Algunas combinaciones no presentan genes por lo que no se tendrán en cuenta.

Ahora se hará un filtrado de los resultados para quedarnos con las combinaciones que tengan un número de genes entre 50 y 300.

```
filtered_results <- results[results$NumGenes >= 50 & results$NumGenes <= 300, ]
filtered_results
```

```
##      LFC COV NumGenes
## 2  1.00  2        51
## 5  1.50  1     287
## 7  1.75  1     172
## 9  2.00  1     114
```

Lo ideal es probar distintos valores de LFC y COV para determinar cuántos genes diferencialmente expresados (DE) se obtienen en cada caso. El objetivo es identificar combinaciones que resulten en un número de genes dentro del rango indicado, es decir, entre 50 y 300 genes.

Por ello, se han evaluado múltiples combinaciones de LFC y COV y se ha generado una tabla con el número de genes DE obtenidos en cada combinación. Finalmente, se selecciona una combinación que

mejor cumpla con el criterio de ser intermedia dentro del rango especificado, asegurando un equilibrio representativo y un análisis manejable. Por tanto, he decidido elegir una cantidad de genes cercana a 175, ya que representa un equilibrio adecuado. Las razones para esta elección son las siguientes:

- Es un valor suficientemente representativo, evitando tanto un número excesivo como insuficiente de genes.
- Facilita un análisis más manejable y reduce el coste computacional.
- Se encuentra en una posición intermedia dentro del rango especificado (50-300 genes).

```
if (nrow(filtered_results) > 0) {  
  best_combination<-filtered_results[which.min(abs(filtered_results$NumGenes-175)),]  
  print(best_combination)  
} else {  
  cat("No se encontraron combinaciones dentro del rango deseado.\n")  
}
```

```
##      LFC COV NumGenes  
## 7 1.75   1      172
```

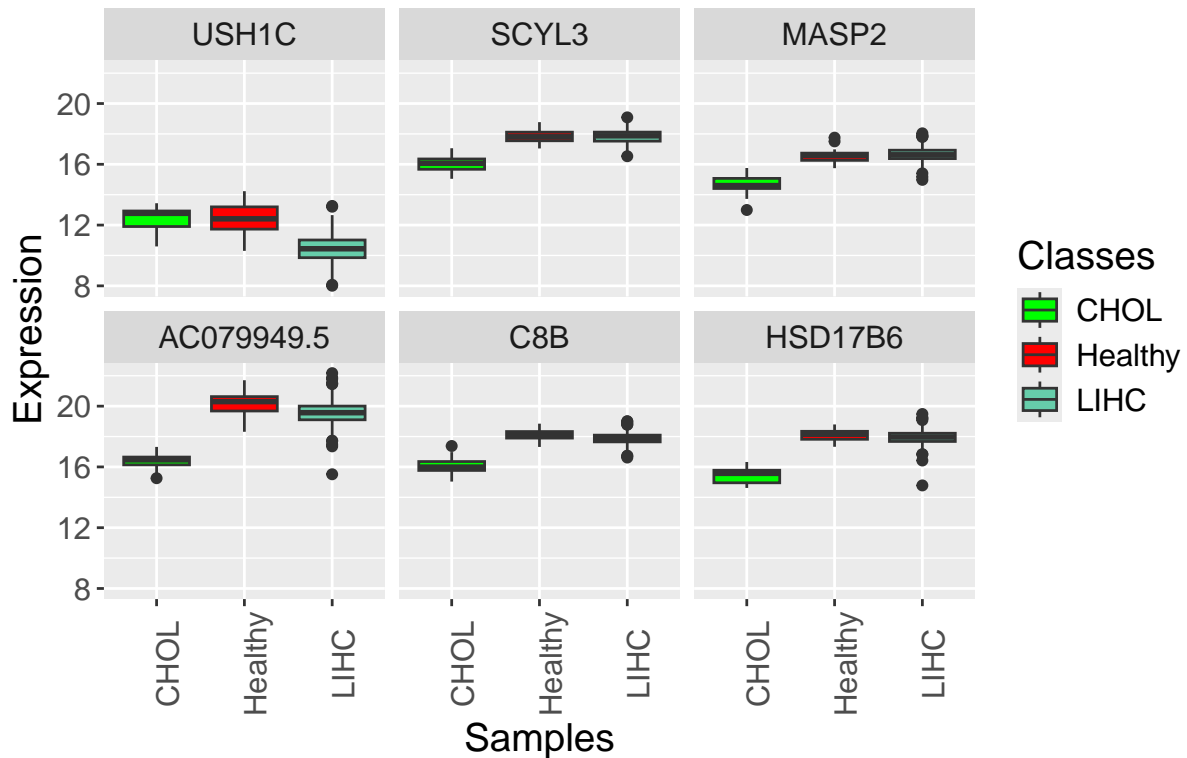
Se ha elegido la combinación con **LFC = 1.75** y **COV = 1.0** ya que es suficientemente representativa, evitando tanto un exceso de genes como la insuficiencia de los mismo.

Por tanto, la mejor combinación para extraer el mejor número de genes es:

- **LFC:** 1.75
- **COV:** 1.0

Una vez que se han elegido los mejores valores de LFC y COV, se mostrará un boxplot de expresión de 6 primeros DEGs.

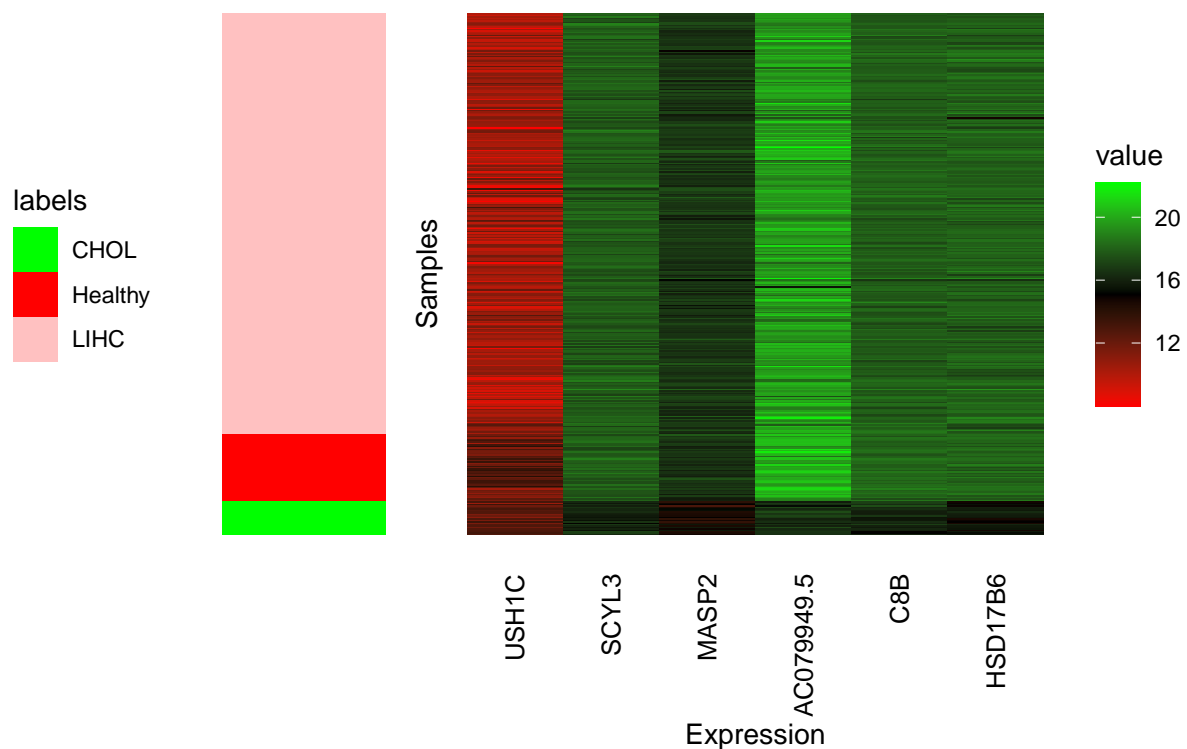
```
dataPlot(DEGsMatrix[1:6, ], YTrn, mode = "genesBoxplot", toPNG=FALSE, toPDF=FALSE)
```



Se observa que, en la mayoría de los genes, existen dos clases que muestran medianas bastante similares entre sí. Por ejemplo, en todos los genes, excepto en AC079949.5, parece que la clase CHOL se distingue adecuadamente de las otras dos clases.

Se realizará el mismo análisis utilizando un heatmap de los primeros 6 genes. Se ha seleccionado un número reducido de 6 genes para facilitar su visualización y evitar la complejidad de interpretar un gran volumen de datos.

```
dataPlot(DEGsMatrix[1:6,], YTrn, mode = "heatmap", toPNG=FALSE, toPDF=FALSE)
```



Gracias a este heatmap, se muestra para cada gen como es cada una de las muestras.

Para finalizar este apartado, se van a separar tanto la matriz como los labels:

```
MLMatrix <- t(DEGsMatrix)
MLLabels <- YTrn
save(MLMatrix,file = "MLMatrix")
save(MLLabels,file = "MLLabels")
```

Una vez que están definidas estas variables, vamos a definir los distintos algoritmos de selección de características que serán utilizados en el siguiente apartado:

```
FSRankingmRMR <- featureSelection(MLMatrix, MLLabels, mode = "mrmr",
                                   vars_selected = colnames(MLMatrix))
FSRankingRF <- featureSelection(MLMatrix, MLLabels, mode = "rf",
                                vars_selected = colnames(MLMatrix))
FSRankingDA <- featureSelection(MLMatrix, MLLabels, mode = "da",
                                vars_selected = colnames(MLMatrix),
                                disease = "LiverCancer")
```

```
# Función FSRankingLasso
FSRankingLasso <- function(MLMatrix, MLLabels) {
```

```

# Convertir MLMatrix a una matriz numérica
MLMatrix <- as.matrix(MLMatrix)

# Verificar si MLLabels es numérico
if (!is.numeric(MLLabels)) {
  stop("MLLabels no es numérico. Asegúrate de que contiene valores numéricos.")
}

# Verificar si MLLabels es constante
if (length(unique(MLLabels)) == 1) {
  stop("MLLabels es constante. Asegúrate de que contiene más de un valor único.")
}

# Realizar la validación cruzada con Lasso
cv_lasso <- cv.glmnet(MLMatrix, MLLabels, alpha=1, nfolds=10, type.measure = "mse",
  standardize = TRUE)

# Mejor valor de lambda encontrado
best_lambda <- cv_lasso$lambda.min

# Ajustar el modelo con el mejor valor de lambda
modelo_lasso <- glmnet(MLMatrix, MLLabels, alpha = 1, lambda = best_lambda,
  standardize = TRUE)

# Obtener los coeficientes del modelo
coef_lasso <- coef(modelo_lasso)

# Convertir coef_lasso a una matriz y extraer coeficientes no nulos
coef_matrix <- as.matrix(coef_lasso)

# Filtrar coeficientes no nulos (excepto el intercepto)
selected_features <- rownames(coef_matrix)[coef_matrix != 0 &
  rownames(coef_matrix) != "(Intercept)"]

# Verifica si hay características seleccionadas
if (length(selected_features) == 0) {
  return(NULL) # Si no se seleccionan características se devuelva NULL
}

```



```
    return(selected_features)
  }

MLLabels <- as.factor(MLLabels)
MLLabels <- as.numeric(MLLabels)

FSRankingLasso <- FSRankingLasso(MLMatrix, MLLabels)
FSRankingLasso <- FSRankingLasso[FSRankingLasso != "(Intercept)"]
```

3 Comparativas algoritmo de selección de características y clasificación

En este apartado se realizarán comparativas entre cuatro algoritmos de selección de características y cuatro algoritmos de clasificación, con el objetivo de identificar cuáles ofrecen los mejores resultados en el análisis de datos. Los algoritmos de selección de características que se evaluarán son los siguientes:

- **mRMR (Minimum Redundancy Maximum Relevance):** Un enfoque basado en la selección de características que maximizan la relevancia y minimizan la redundancia entre ellas.
- **RF (Random Forest):** Un algoritmo basado en árboles de decisión que puede utilizarse tanto para clasificación como para la selección de características, evaluando la importancia de cada una.
- **DA (Discriminant Analysis):** Un método estadístico que selecciona las características más relevantes para maximizar la separación entre las clases en el conjunto de datos.
- **Lasso (Least Absolute Shrinkage and Selection Operator):** Un método de regularización que selecciona características mediante una penalización en los coeficientes del modelo, llevando a cero aquellos que no son significativos.

Estos algoritmos de selección de características se evaluarán en combinación con cuatro algoritmos de clasificación para determinar cuál de ellos ofrece el mejor rendimiento en este análisis:

- **k-NN (k-Nearest Neighbors):** Un clasificador basado en la proximidad de los datos en el espacio de características.
- **Random Forest:** Un clasificador que utiliza múltiples árboles de decisión para hacer predicciones robustas, basado en la agregación de decisiones de varios modelos.
- **SVM (Support Vector Machines):** Un clasificador que busca la frontera óptima para separar las clases en el espacio de características, maximizando el margen entre ellas.
- **Redes Neuronales:** Modelos de clasificación que simulan el comportamiento del cerebro humano mediante una estructura de neuronas artificiales, capaces de aprender patrones complejos a partir de los datos.

El rendimiento de cada combinación de algoritmo de selección de características y clasificador se evaluará a través de diferentes métricas, generando gráficas para facilitar la comparación entre los métodos. Este análisis permitirá identificar qué combinación de algoritmos proporciona los mejores resultados en términos de precisión y capacidad predictiva.

3.1 Algoritmo de clasificación: k-NN

En este apartado se probarán distintos algoritmos de selección de características para el algoritmo, como se ha comentado anteriormente son cuatro.

3.1.1 Algoritmo de selección de características: mRMR

En primer lugar, realizamos el entrenamiento del modelo k-NN utilizando las 10 características seleccionadas por el método **mRMR**:

```
knn_trn_mrmr <- knn_trn(MLMatrix, MLLabels, vars_selected = FSRankingmRMR[1:10])
```

```
save(knn_trn_mrmr, file = 'knn_trn_mrmr')
```

```
load(file = "knn_trn_mrmr")
```

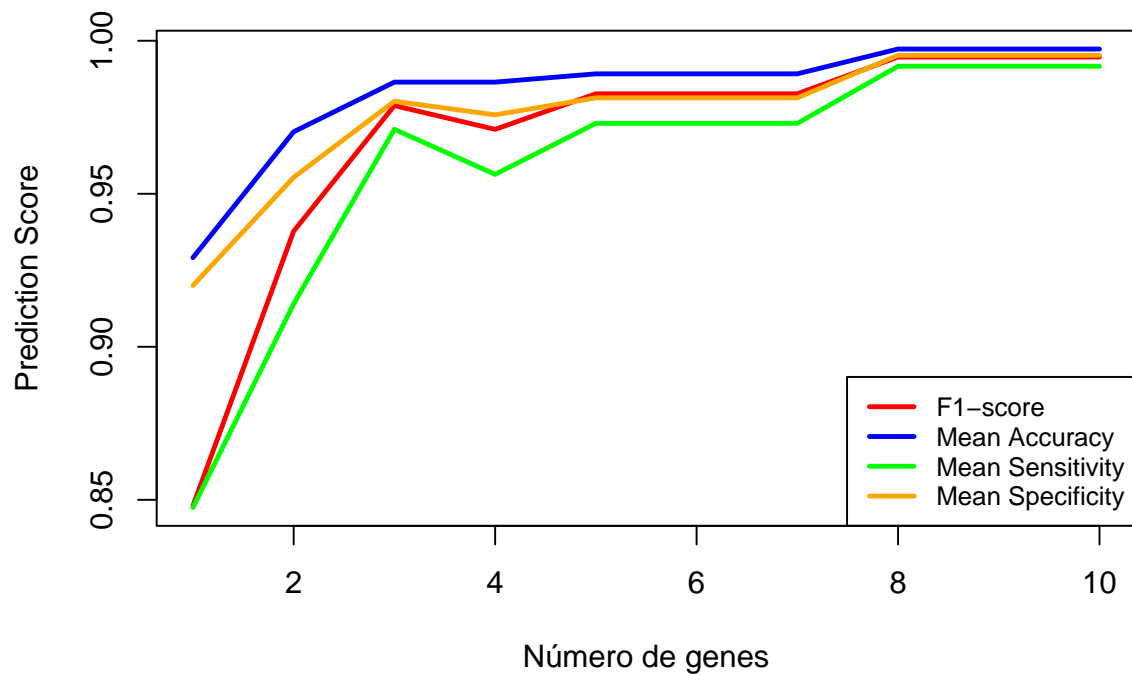
Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

```
knn_results_mrmr_trn <- rbind(knn_trn_mrmr$F1Info$meanF1[1:10],  
  knn_trn_mrmr$accuracyInfo$meanAccuracy,knn_trn_mrmr$sensitivityInfo$meanSensitivity,  
  knn_trn_mrmr$specificityInfo$meanSpecificity)  
save(knn_results_mrmr_trn,file ="knn_results_mrmr_trn" )
```

```
load(file = "knn_results_mrmr_trn")
```

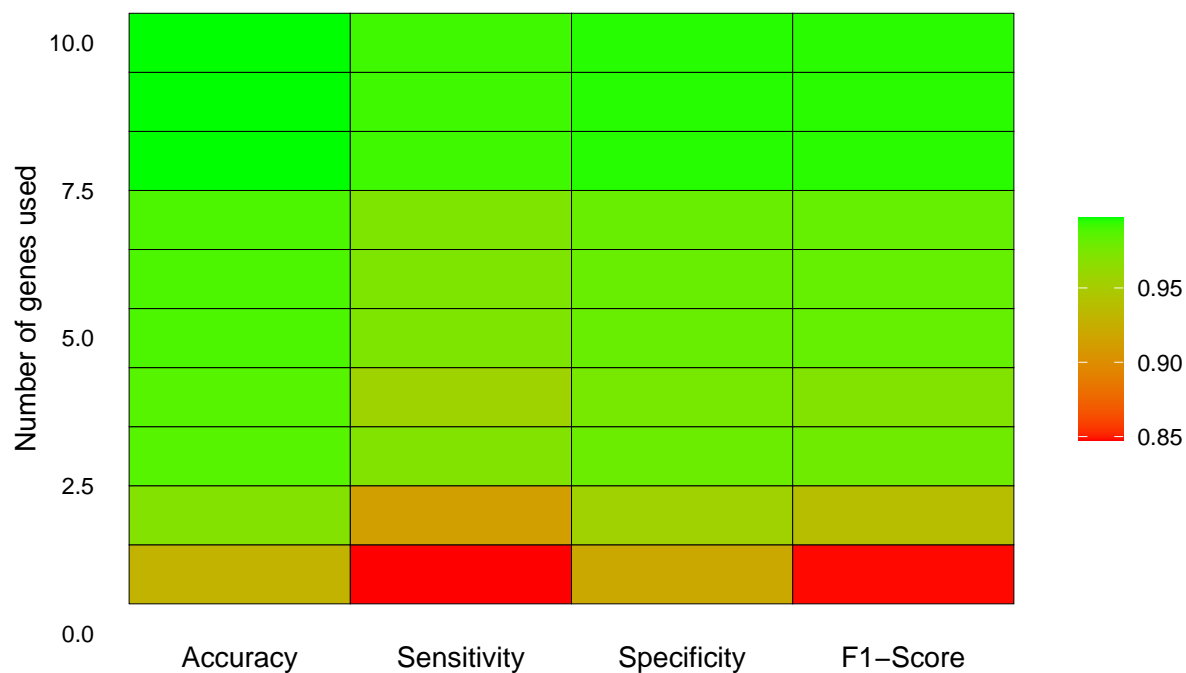
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(knn_results_mrmr_trn, MLLabels, legend = c("F1-score","Mean Accuracy",  
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",  
  xlab="Número de genes", ylab="Prediction Score")
```

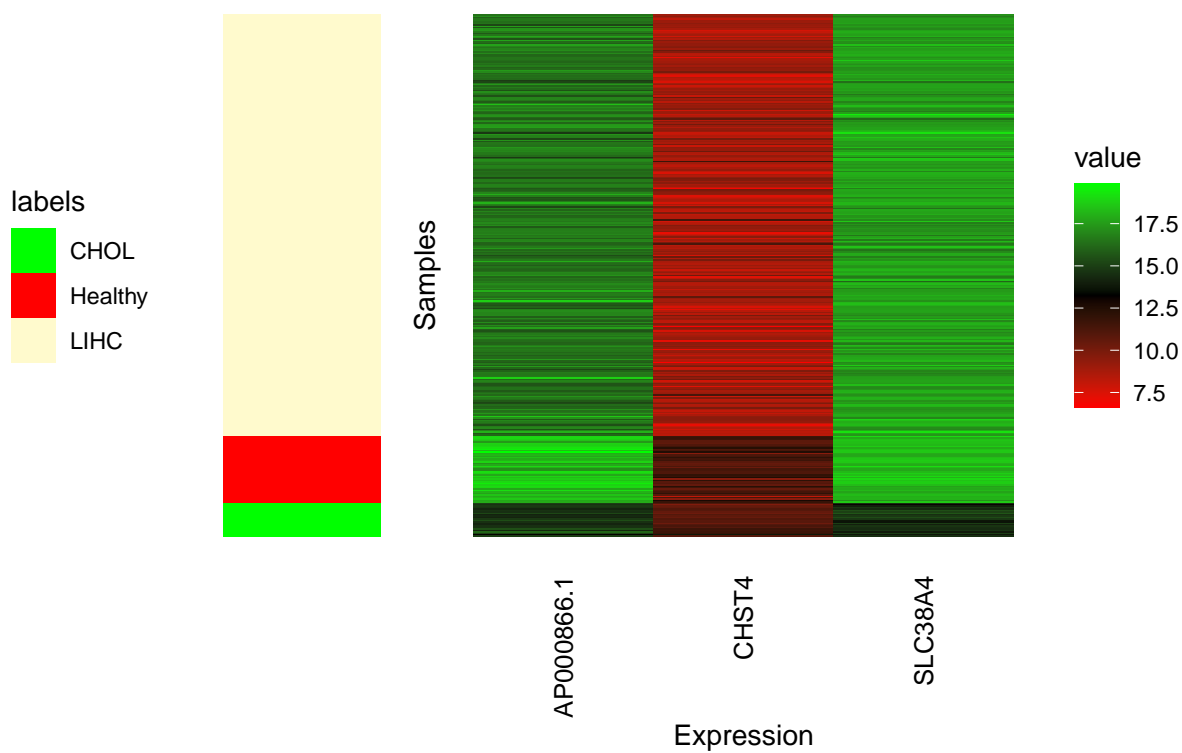


Mostramos un mapa de calor de los resultados y los valores correspondientes a las tres características principales seleccionadas, **este último gráfico solo se mostrará una vez ya que se guardó en una variable el FSRankingmRMR y por tanto siempre tiene el mismo valor, lo mismo ocurre para el resto de rankings:**

```
dataPlot(knn_trn_mrmr, MLLabels, mode = "heatmapResults")
```



```
dataPlot(t(MLMatrix[,names(FSRankingmRMR[1:3]))), MLLabels, mode = "heatmap")
```



Aunque lo normal es evaluar las métricas **Accuracy** y **F1-score**, hemos añadido también **Sensitivity**

y **Specificity** en el entrenamiento para un análisis más completo. Además, se presentan varias gráficas que permiten analizar en mayor profundidad el comportamiento de las características seleccionadas.

A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por mRMR y el valor de **k** obtenido durante el entrenamiento:

```
knn_test_mrmr <- knn_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
                        FSRankingmRMR[1:10], bestK=knn_trn_mrmr$bestK)
save(knn_test_mrmr, file = "knn_test_mrmr")
```

```
load(file = "knn_test_mrmr")
```

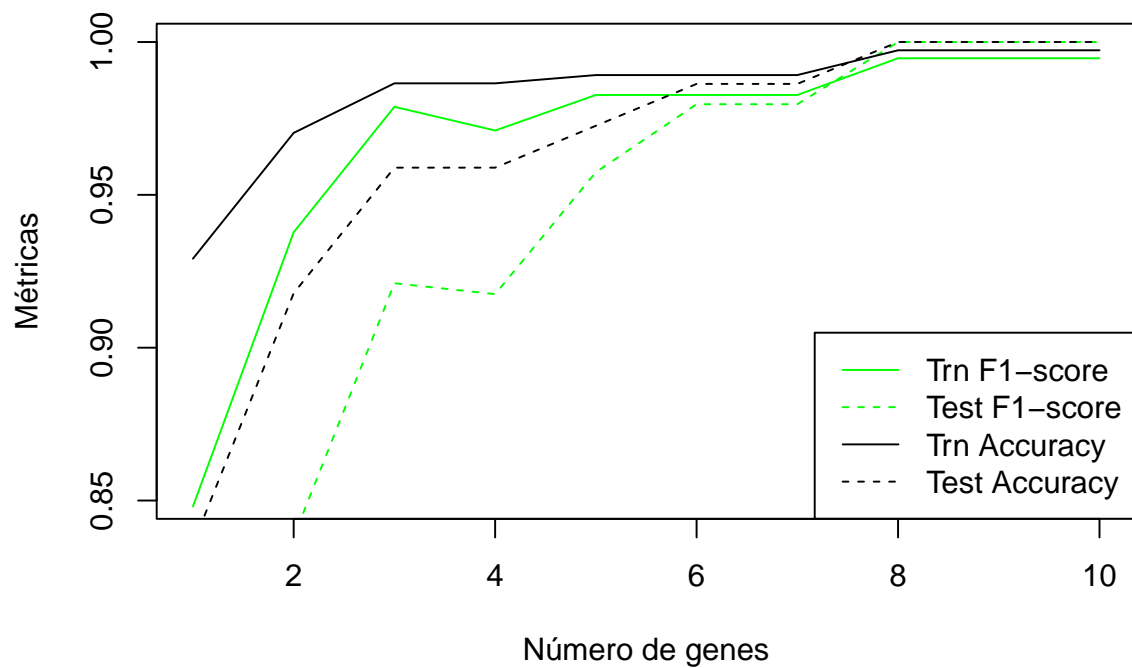
Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
knn_results_mrmr_test <- rbind(knn_trn_mrmr$F1Info$meanF1[1:10],knn_test_mrmr$f1Vector,
                              knn_trn_mrmr$accuracyInfo$meanAccuracy, knn_test_mrmr$accVector)
save(knn_results_mrmr_test, file = "knn_results_mrmr_test")
```

```
load(file = "knn_results_mrmr_test")
```

Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- knn_results_mrmr_test[1,1:10]
Test_F1 <- knn_results_mrmr_test[2,1:10]
Trn_Acc <- knn_results_mrmr_test[3,1:10]
Test_Acc <- knn_results_mrmr_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.85,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )
```



Mostramos la matriz de confusión en el conjunto de test:

```
dataPlot(knn_test_mrmr$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")
```

Reference	CHOL	4	0	0
	Healthy	0	9	1
	LIHC	1	1	57
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
95.89	92.108	95.537	96.607

La gráfica parece indicar que los datos de test van mejor que los datos de entrenamiento. Según lo indicado por los tutores, hay que escoger a partir del TRN. Con **tres genes** parece que los TRN tienen un pico, por lo que he decidido coger ese número de genes para este apartado.

3.1.2 Algoritmo de selección de características: RF

En esta sección se utilizará el **método de selección de características Random Forest (RF)**. Se seguirán los mismos pasos aplicados anteriormente con mRMR para analizar los resultados obtenidos tanto en el entrenamiento como en el test, y así comparar su rendimiento.

En primer lugar, realizamos el entrenamiento del modelo k-NN utilizando las 10 características seleccionadas por el método **RF**:

```
knn_trn_rf <- knn_trn(MLMatrix, MLLabels, vars_selected = FSRankingRF[1:10])  
save(knn_trn_rf, file = "knn_trn_rf" )
```

```
load(file = "knn_trn_rf")
```

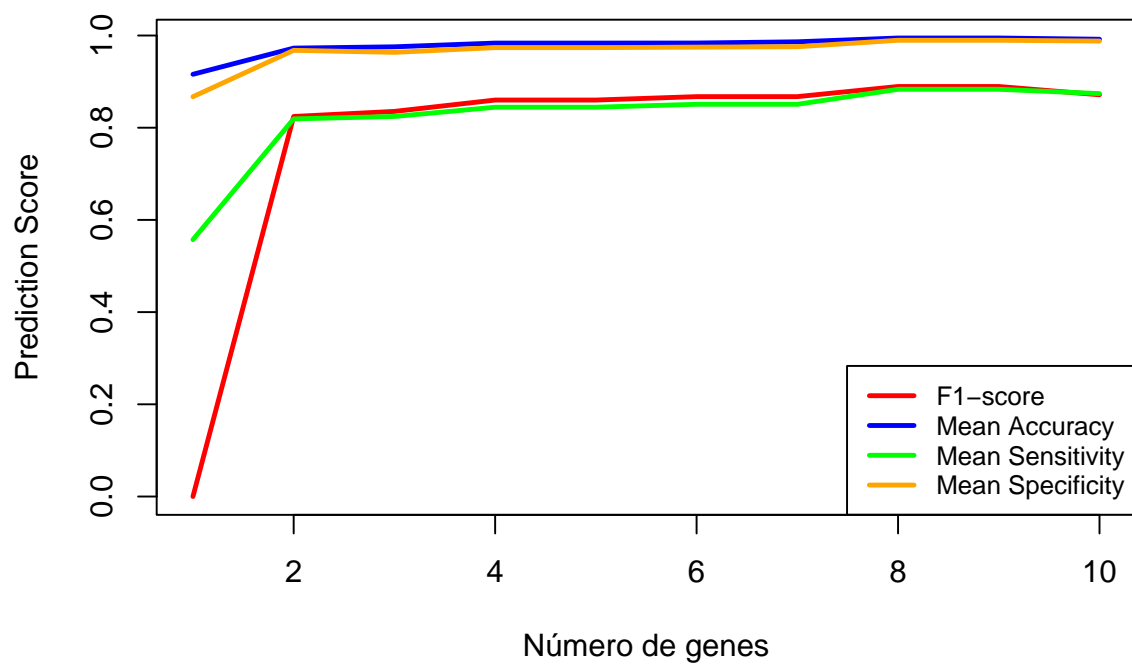
Se combinan las métricas principales (F1-score, Accuracy, Sensitivity y Specificity) para los 10 primeros genes seleccionados por RF en una sola matriz (knn_results_rf_trn), lo que facilita su visualización.

```
knn_results_rf_trn <- rbind(knn_trn_rf$F1Info$meanF1[1:10],  
  knn_trn_rf$accuracyInfo$meanAccuracy, knn_trn_rf$sensitivityInfo$meanSensitivity,  
  knn_trn_rf$specificityInfo$meanSpecificity)  
save(knn_results_rf_trn, file = "knn_results_rf_trn")
```

```
load(file = "knn_results_rf_trn")
```

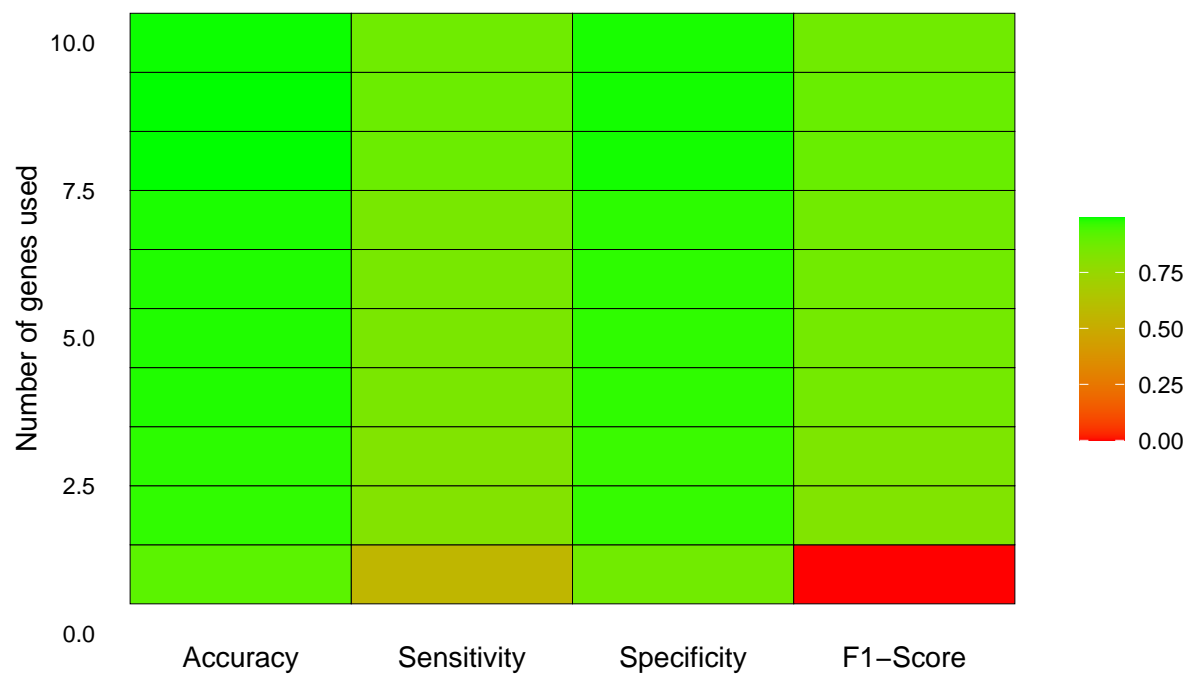
A continuación, se crea una gráfica que muestre el rendimiento de cada métrica en función del número de genes.

```
dataPlot(knn_results_rf_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",  
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",  
  xlab="Número de genes", ylab="Prediction Score")
```

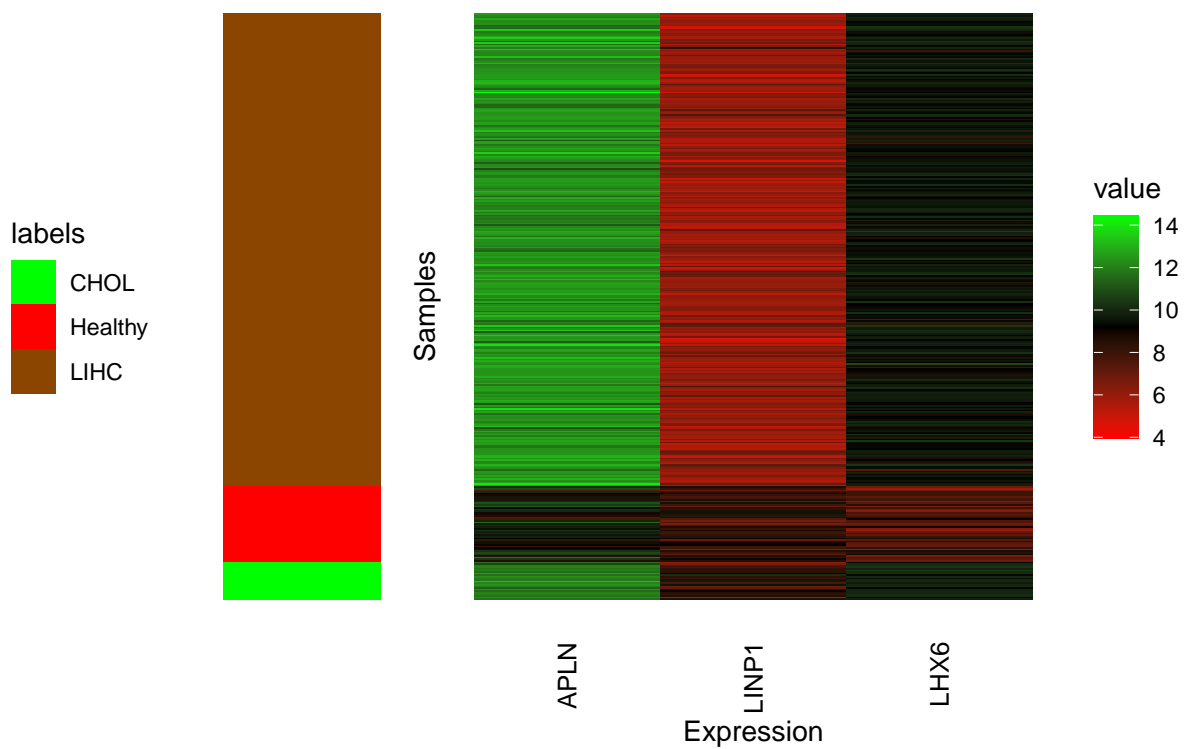



Se mostrará un mapa de calor con los resultados globales del entrenamiento, al igual que se visualizará la expresión de los tres genes más importantes, seleccionados por RF:

```
dataPlot(knn_trn_rf, MLLabels, mode = "heatmapResults")
```



```
dataPlot(t(MLMatrix[,FSRankingRF[1:3]]), MLLabels, mode = "heatmap")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas

por RF y el valor de k obtenido durante el entrenamiento:

```
knn_test_rf <- knn_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
                      FSRankingRF[1:10], bestK=knn_trn_rf$bestK)
save(knn_test_rf, file = "knn_test_rf")
```

```
load(file = "knn_test_rf")
```

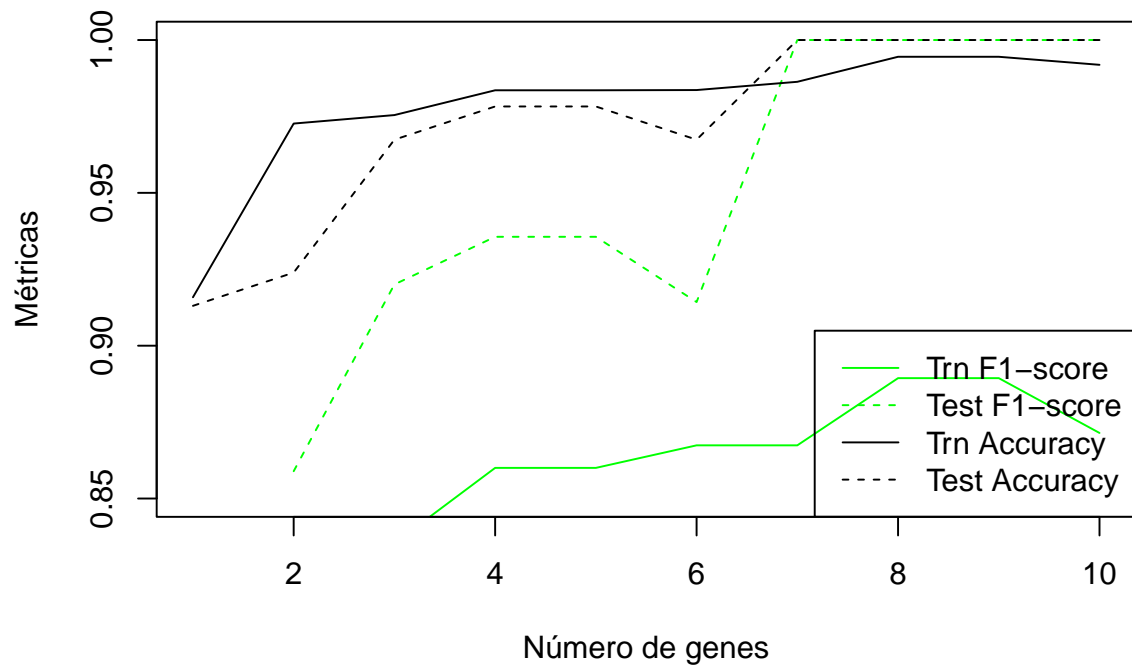
Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
knn_results_rf_test <- rbind(knn_trn_rf$F1Info$meanF1[1:10], knn_test_rf$f1Vector,
                             knn_trn_rf$accuracyInfo$meanAccuracy, knn_test_rf$accVector)
save(knn_results_rf_test, file = "knn_results_rf_test")
```

```
load(file = "knn_results_rf_test")
```

Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- knn_results_rf_test[1,1:10]
Test_F1 <- knn_results_rf_test[2,1:10]
Trn_Acc <- knn_results_rf_test[3,1:10]
Test_Acc <- knn_results_rf_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.85,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )
```



Mostramos la matriz de confusión en el conjunto de test:

```
dataPlot(knn_test_rf$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")
```

Reference	CHOL	6	1	0
	Healthy	0	11	0
	LIHC	1	1	72
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
96.739	92.004	94.337	98.785

En la primera gráfica, que muestra la predicción de los scores en función del número de genes, se observa que con cuatro genes se puede obtener una predicción bastante precisa, al igual que con ocho. La elección del número de genes dependerá de si se prefiere utilizar un conjunto más amplio de genes o trabajar con un número reducido. En mi opinión, la mejor opción sería utilizar **cuatro genes**.

3.1.3 Algoritmo de selección de características: DA

Esta una de las últimas pruebas que se hará para el algoritmo k-NN, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo k-NN utilizando las 10 características seleccionadas por el método **DA**:

```
knn_trn_da <- knn_trn(MLMatrix, MLLabels, vars_selected = names(FSRankingDA[1:10]))
save(knn_trn_da, file = "knn_trn_da")
```

```
load(file = "knn_trn_da")
```

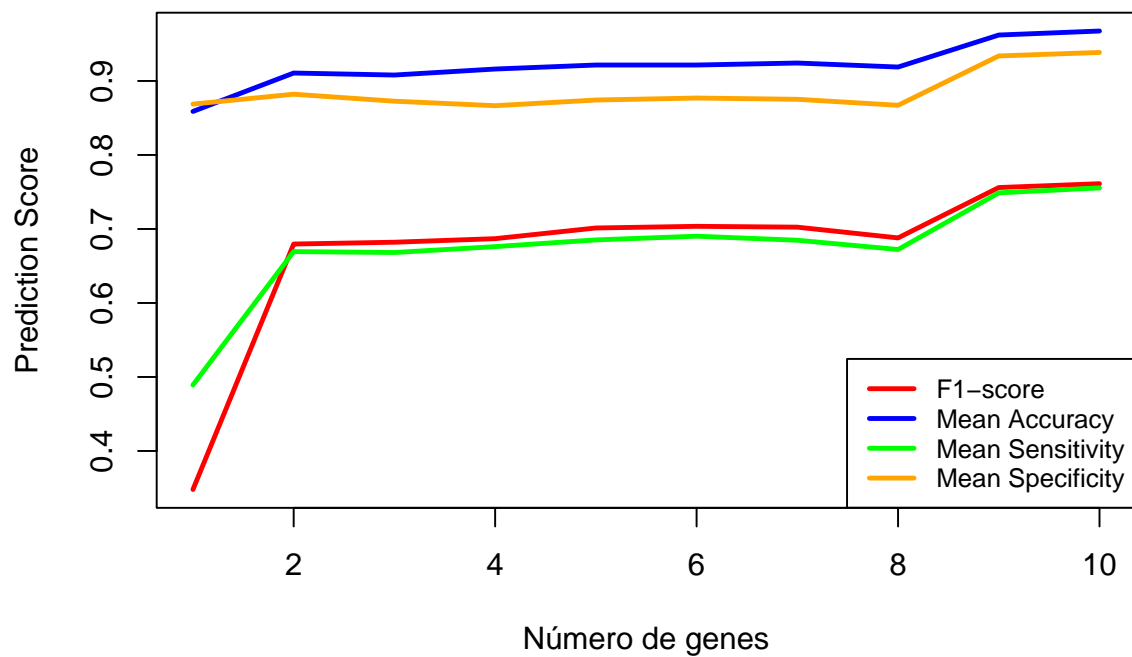
Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

```
knn_results_da_trn <- rbind(knn_trn_da$F1Info$meanF1[1:10],
  knn_trn_da$accuracyInfo$meanAccuracy, knn_trn_da$sensitivityInfo$meanSensitivity,
  knn_trn_da$specificityInfo$meanSpecificity)
save(knn_results_da_trn, file = "knn_results_da_trn")
```

```
load(file = "knn_results_da_trn")
```

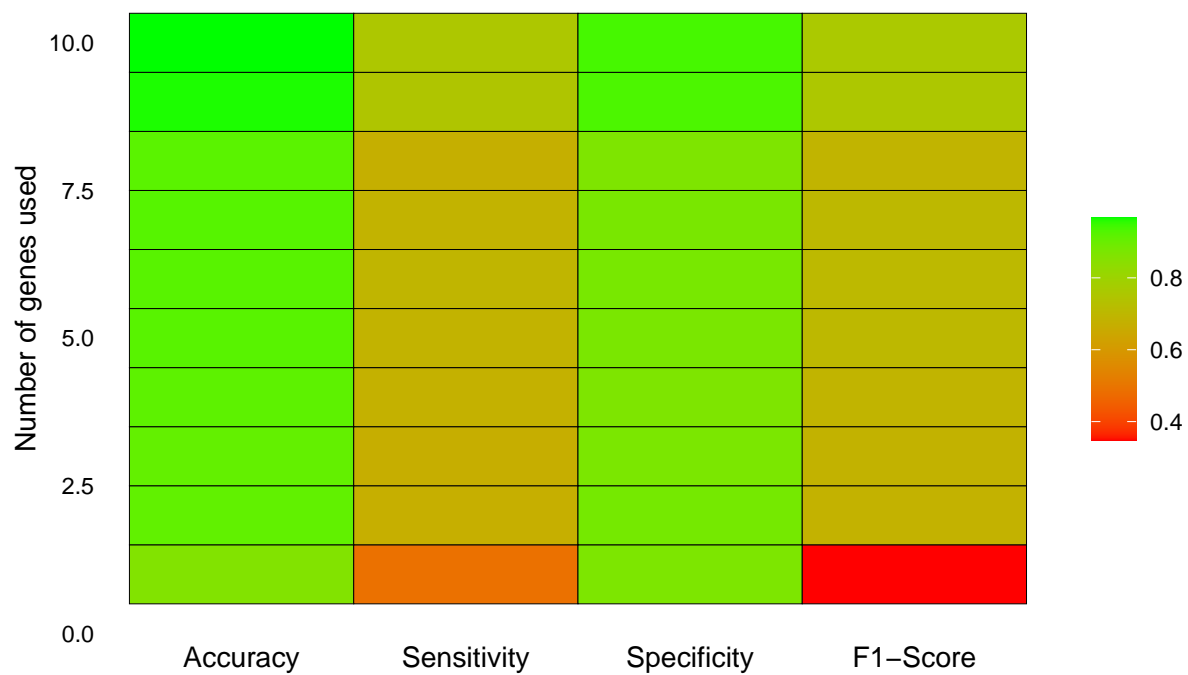
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(knn_results_da_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```

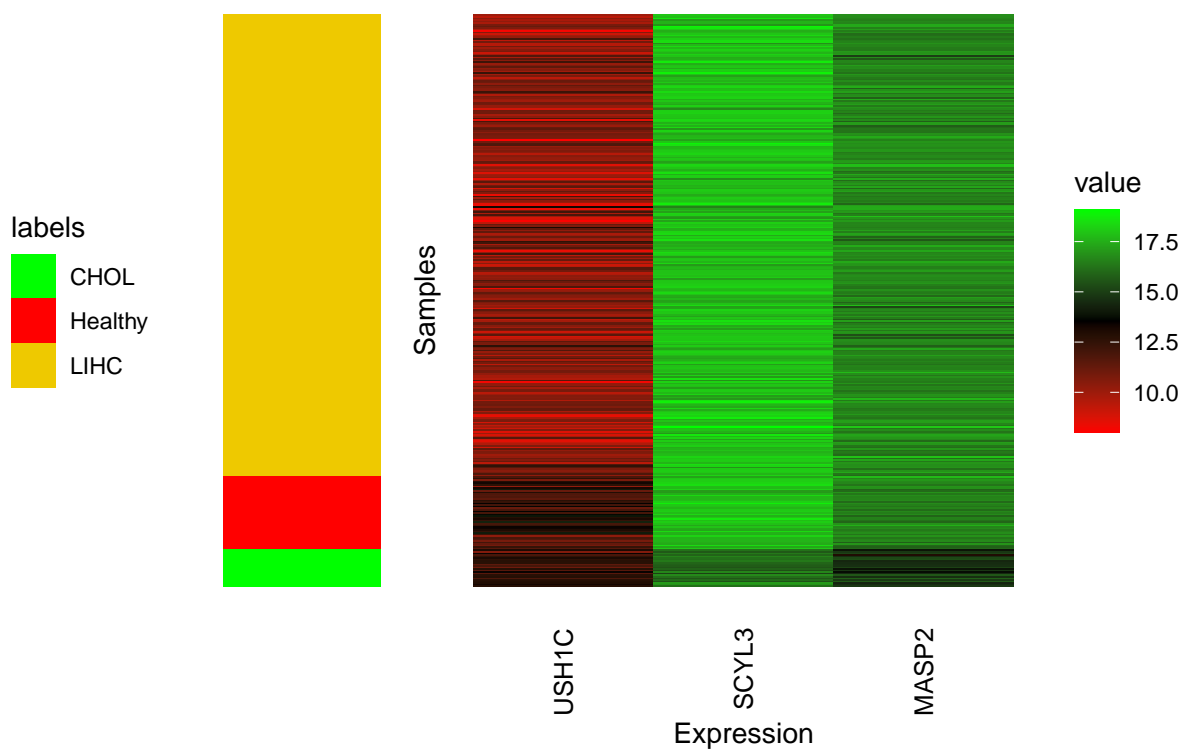


Mostramos un mapa de calor de los resultados y los valores correspondientes a las tres características principales seleccionadas:

```
dataPlot(knn_trn_da, MLLabels, mode = "heatmapResults")
```



```
dataPlot(t(MLMatrix[,names(FSRankingDA[1:3])]), MLLabels, mode = "heatmap")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas

por DA y el valor de k obtenido durante el entrenamiento:

```
knn_test_da <- knn_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
                        names(FSRankingDA[1:10]), bestK=knn_trn_da$bestK)
save(knn_test_da,file = "knn_test_da")
```

```
load(file = "knn_test_da")
```

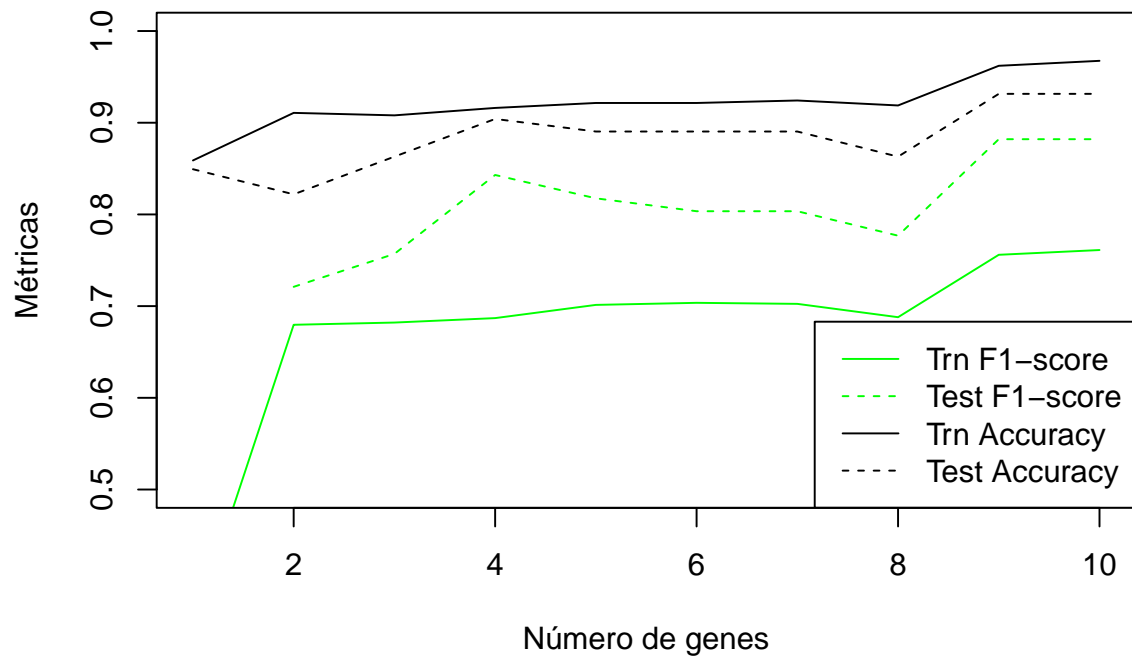
Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
knn_results_da_test <- rbind(knn_trn_da$F1Info$meanF1[1:10], knn_test_da$f1Vector,
                             knn_trn_da$accuracyInfo$meanAccuracy, knn_test_da$accVector)
save(knn_results_da_test,file = "knn_results_da_test" )
```

```
load(file = "knn_results_da_test")
```

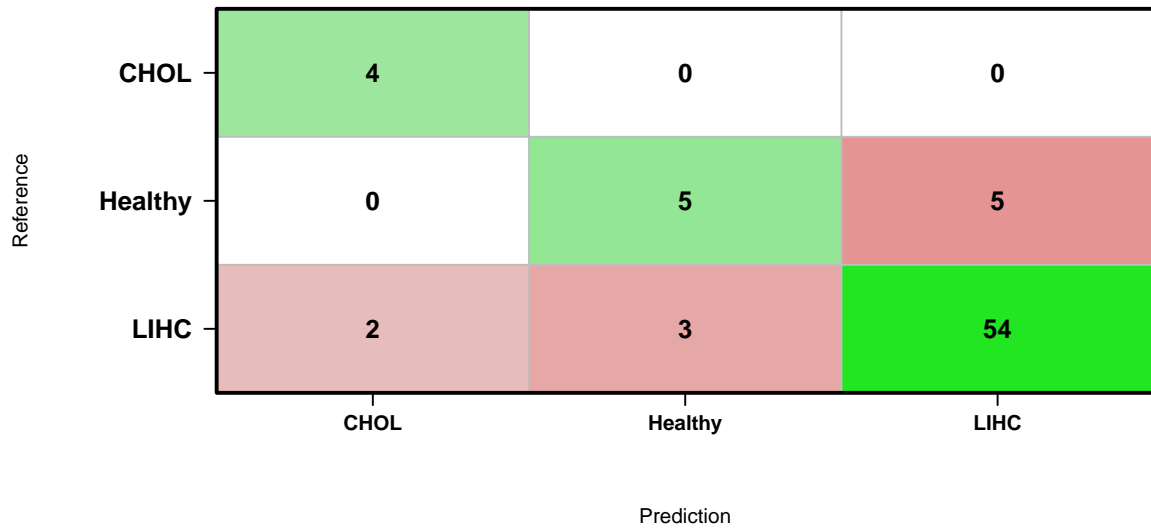
Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- knn_results_da_test[1,1:10]
Test_F1 <- knn_results_da_test[2,1:10]
Trn_Acc <- knn_results_da_test[3,1:10]
Test_Acc <- knn_results_da_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.5,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )
```

Mostramos la matriz de confusión en el conjunto de test:

```
dataPlot(knn_test_da$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")
```



DETAILS

Accuracy	F1	Sensitivity	Specificity
86.301	75.694	80.508	85.542

En esta gráfica podemos observar varios picos, dependiendo del número de genes utilizados. En mi opinión, la opción más adecuada sería emplear **nueve genes**, ya que parece ser la opción que ofrece más explicabilidad.

3.1.4 Algoritmo de selección de características: Lasso

Esta será la última prueba que se hará para el algoritmo k-NN, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo k-NN utilizando las 10 características seleccionadas por el método **Lasso**:

```
knn_trn_lasso <- knn_trn(MLMatrix, MLLabels, vars_selected = FSRankingLasso[1:10])  
save(knn_trn_lasso, file = "knn_trn_lasso")
```

```
load(file = "knn_trn_lasso")
```

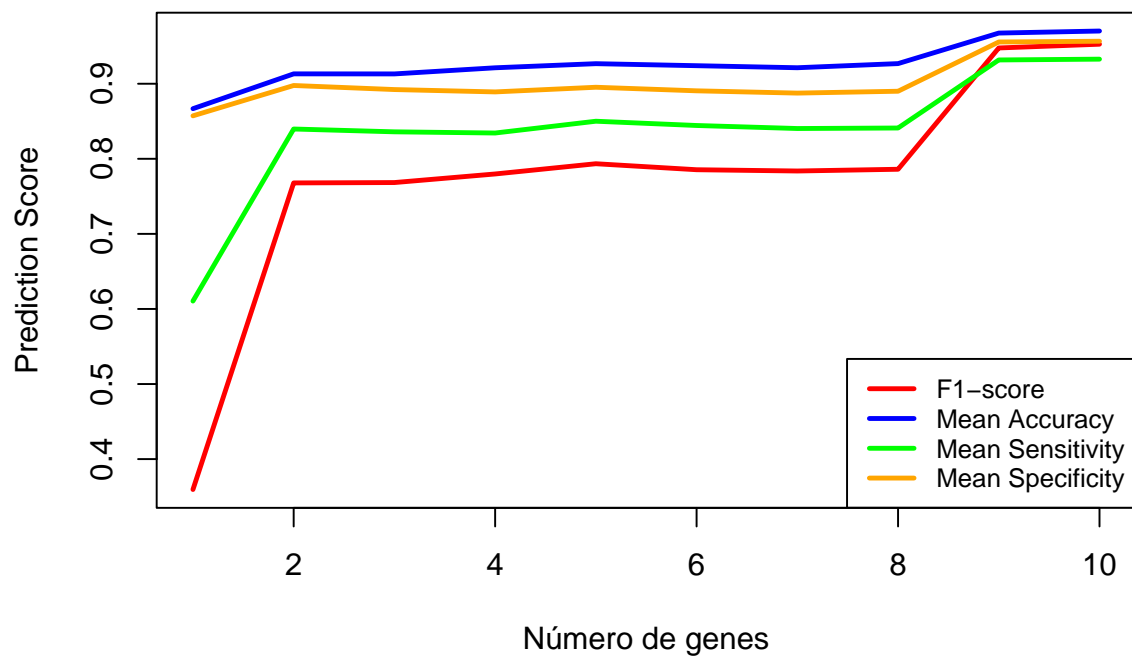
Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

```
knn_results_lasso_trn <- rbind(knn_trn_lasso$F1Info$meanF1[1:10],  
  knn_trn_lasso$accuracyInfo$meanAccuracy,  
  knn_trn_lasso$sensitivityInfo$meanSensitivity,  
  knn_trn_lasso$specificityInfo$meanSpecificity)  
save(knn_results_lasso_trn, file = "knn_results_lasso_trn")
```

```
load(file = "knn_results_lasso_trn")
```

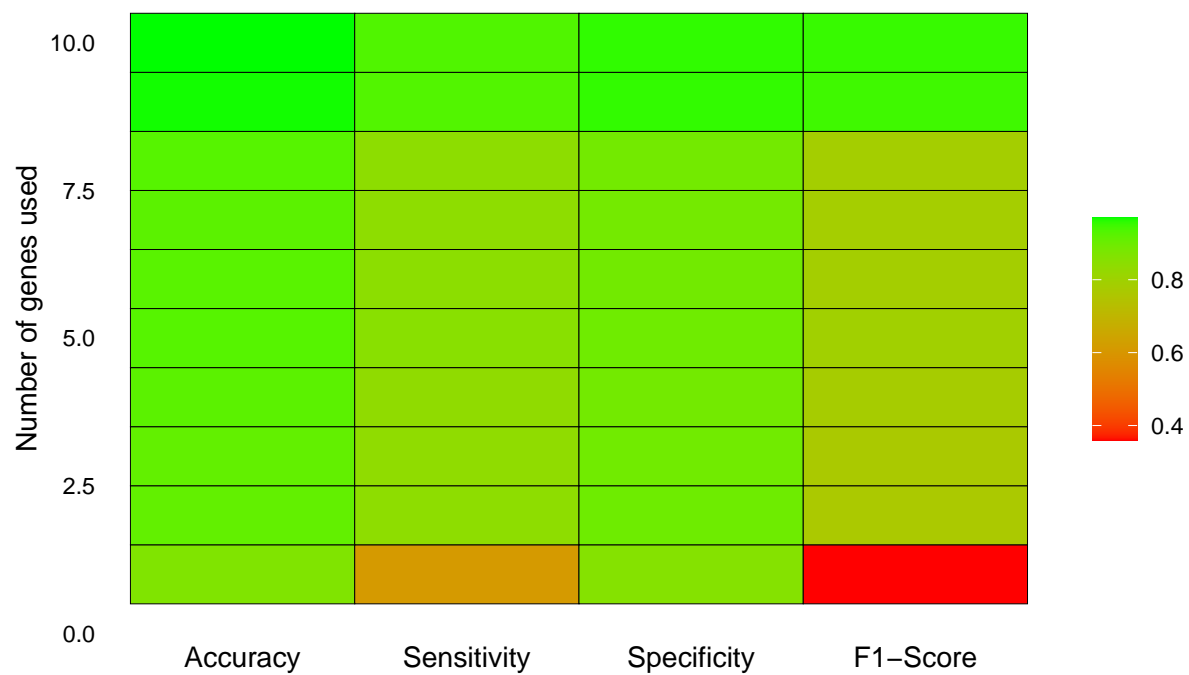
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(knn_results_lasso_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",  
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",  
  xlab="Número de genes", ylab="Prediction Score")
```

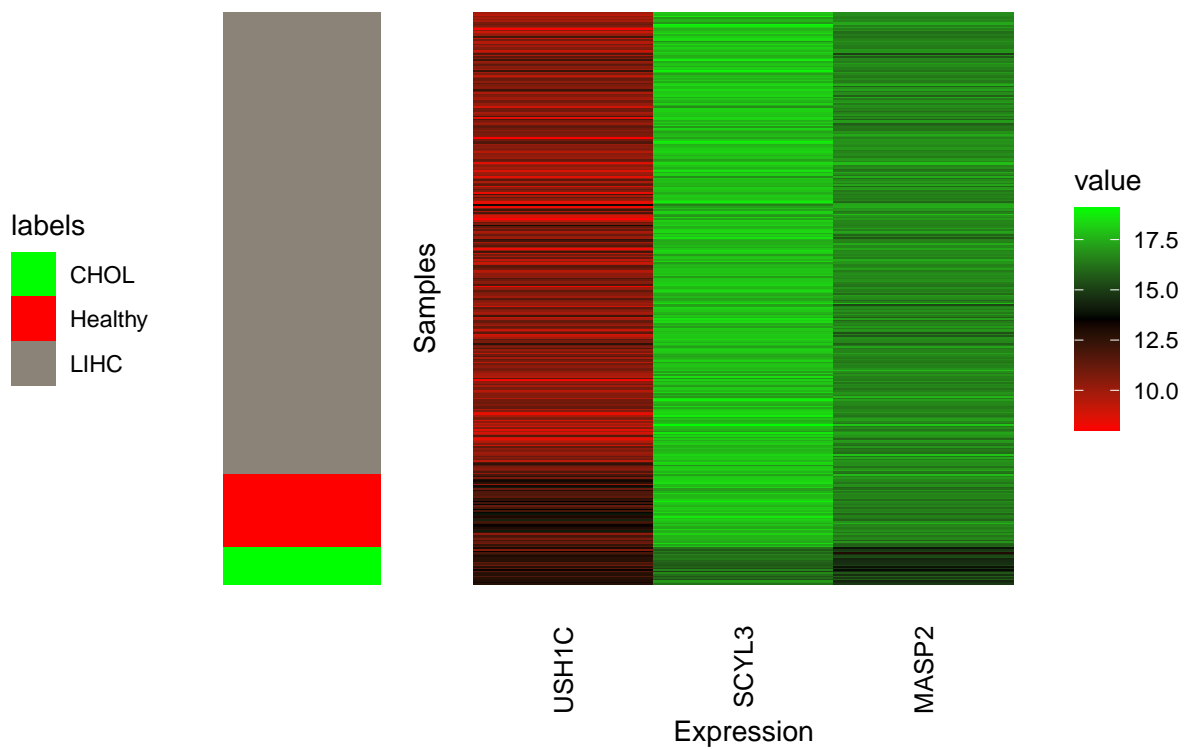


Mostramos un mapa de calor de los resultados y los valores correspondientes a las tres características principales seleccionadas:

```
dataPlot(knn_trn_lasso, MLLabels, mode = "heatmapResults")
```



```
dataPlot(t(MLMatrix[,FSRankingLasso[1:3]]), MLLabels, mode = "heatmap")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas

por Lasso y el valor de k obtenido durante el entrenamiento:

```
knn_test_lasso <- knn_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
                        FSRankingLasso[1:10], bestK=knn_trn_lasso$bestK)
save(knn_test_lasso,file = "knn_test_lasso")
```

```
load(file = "knn_test_lasso")
```

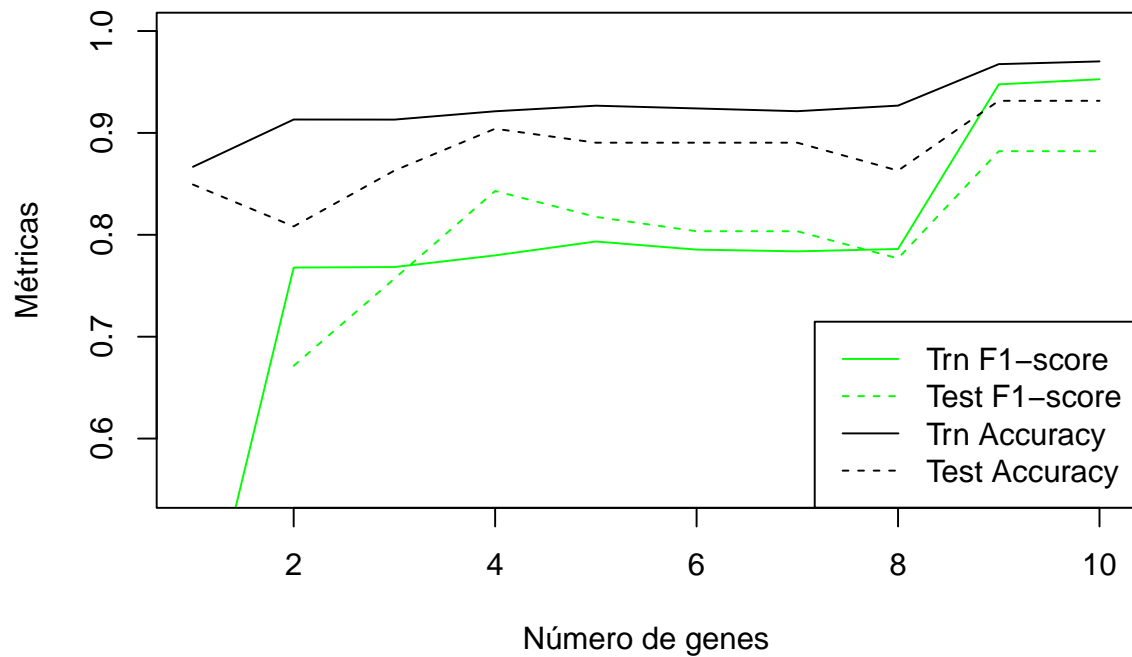
Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
knn_results_lasso_test<-rbind(knn_trn_lasso$F1Info$meanF1[1:10],
    knn_test_lasso$f1Vector,knn_trn_lasso$accuracyInfo$meanAccuracy,
    knn_test_lasso$accVector)
save(knn_results_lasso_test,file = "knn_results_lasso_test" )
```

```
load(file = "knn_results_lasso_test")
```

Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- knn_results_lasso_test[1,1:10]
Test_F1 <- knn_results_lasso_test[2,1:10]
Trn_Acc <- knn_results_lasso_test[3,1:10]
Test_Acc <- knn_results_lasso_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.55,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score", "Test F1-score", "Trn Accuracy", "Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )
```



Mostramos la matriz de confusión en el conjunto de test:

```
dataPlot(knn_test_lasso$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")
```

Reference	CHOL	4	0	0
	Healthy	0	5	5
	LIHC	2	3	54
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
86.301	75.694	80.508	85.542

En esta gráfica podemos observar varios picos, dependiendo del número de genes utilizados. En mi opinión, la opción más adecuada sería emplear nueve genes, ya que parece ofrecer una buena explicabilidad con ese número. Aunque las métricas de TRN también parecen tener un pico con dos genes. Sin embargo, creo que elegiré **9 genes** ya que se muestra un pico superior en ambas métricas.

3.1.5 Conclusiones

Para el **algoritmo k-NN**, tras analizar los resultados obtenidos con diferentes métodos de selección de características, considero que la mejor opción es utilizar **Random Forest** con un número de **4 genes** seleccionados. Con esta configuración, se logra superar el 90% de predicción, alcanzando además los valores más altos de **Accuracy** y **F1-score** en comparación con los demás algoritmos de selección de características, mirando la matriz de confusión correspondiente.

3.2 Algoritmo de clasificación: Random Forest

A continuación, se hará lo mismo que en el apartado anterior, es decir, probar distintos métodos de selección de características y quedarnos con el mejor, pero esta vez para el algoritmo Random Forest.

3.2.1 Algoritmo de selección de características: mRMR

En primer lugar, realizamos el entrenamiento del modelo Random Forest utilizando las 10 características seleccionadas por el método mRMR:

```
rf_trn_mrmr <- rf_trn(MLMatrix, MLLabels, vars_selected = FSRankingmRMR[1:10], numFold = 5)
save(rf_trn_mrmr, file = "rf_trn_mrmr")
```

```
load(file = "rf_trn_mrmr")
```

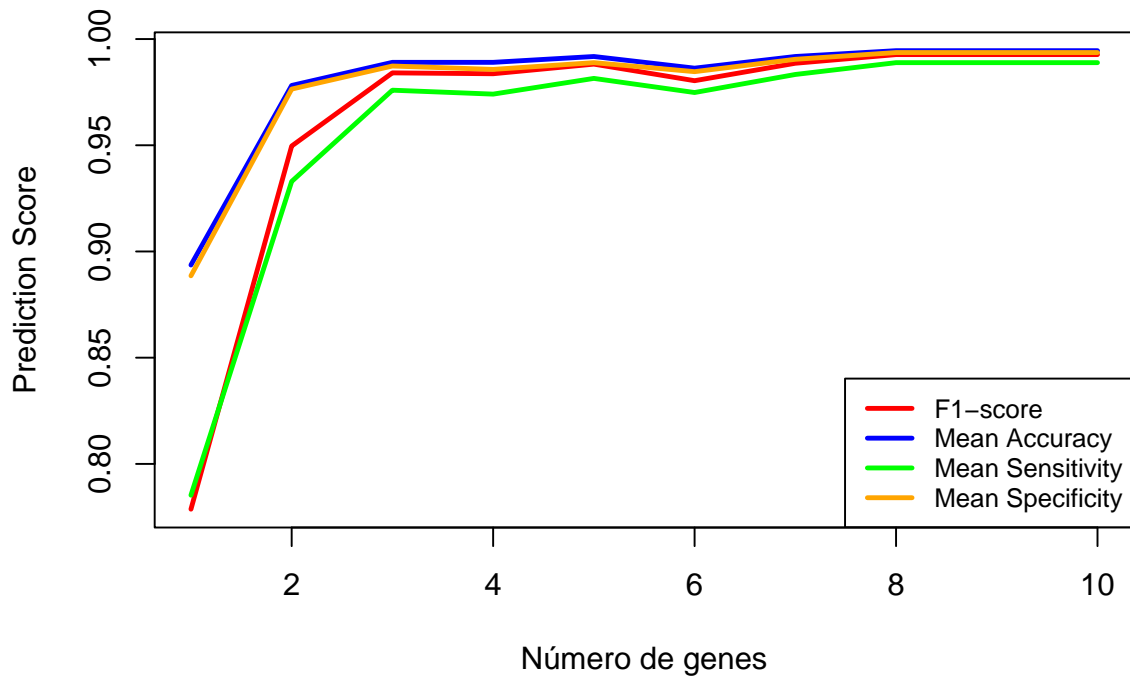
Calculamos las métricas de evaluación **F1-score**, **Accuracy**, **Sensitivity** y **Specificity** y almacenamos los resultados:

```
rf_results_mrmr_trn <- rbind(rf_trn_mrmr$F1Info$meanF1[1:10],
  rf_trn_mrmr$accuracyInfo$meanAccuracy, rf_trn_mrmr$sensitivityInfo$meanSensitivity,
  rf_trn_mrmr$specificityInfo$meanSpecificity)
save(rf_results_mrmr_trn, file = "rf_results_mrmr_trn")
```

```
load(file = "rf_results_mrmr_trn" )
```

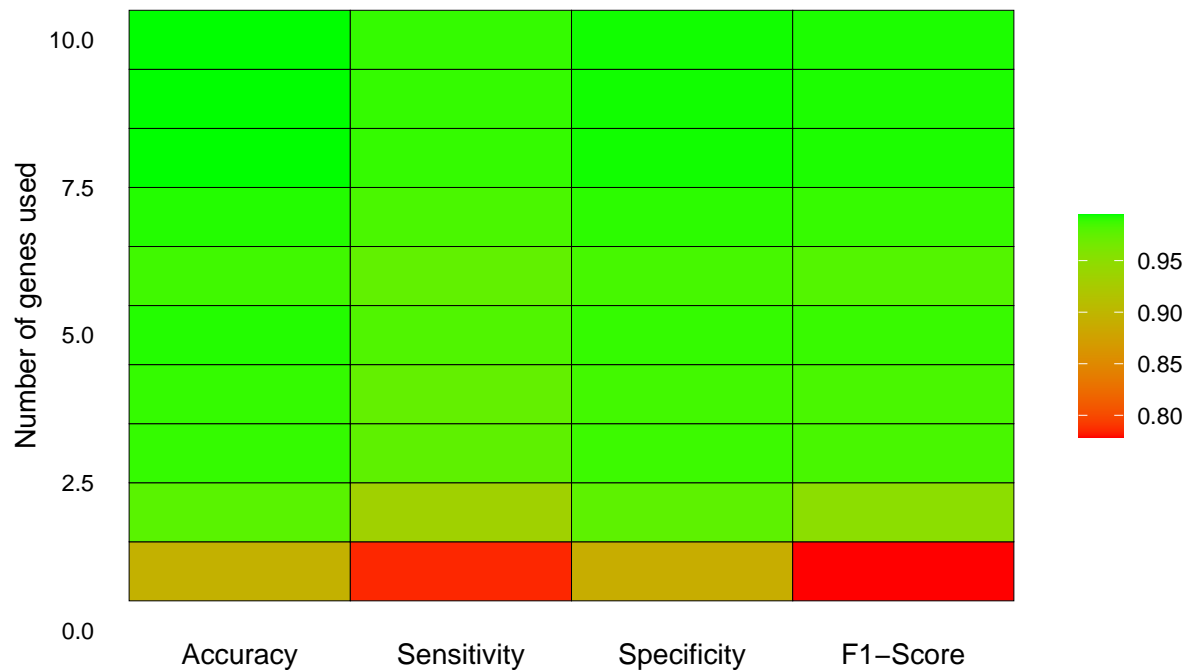
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(rf_results_mrmr_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados, en este caso, ya no se mostrarán un heatmap mostrando los valores correspondientes a las tres características principales ya que se mostraron en el apartado anterior con el algoritmo de clasificación k-NN y se obtiene el mismo resultado:

```
dataPlot(rf_trn_mrmr, MLLabels, mode = "heatmapResults")
```

A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por mRMR y el los best params obtenido durante el entrenamiento:

```
rf_test_mrmr <- rf_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
  FSRankingmRMR[1:10], bestParameters=rf_trn_mrmr$bestParameters)
save(rf_test_mrmr, file = "rf_test_mrmr")
```

```
load(file = "rf_test_mrmr")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
rf_results_mrmr_test <- rbind(rf_trn_mrmr$F1Info$meanF1[1:10], rf_test_mrmr$f1Vector,
  rf_trn_mrmr$accuracyInfo$meanAccuracy, rf_test_mrmr$accVector)
save(rf_results_mrmr_test, file = "rf_results_mrmr_test")
```

```
load(file = "rf_results_mrmr_test")
```

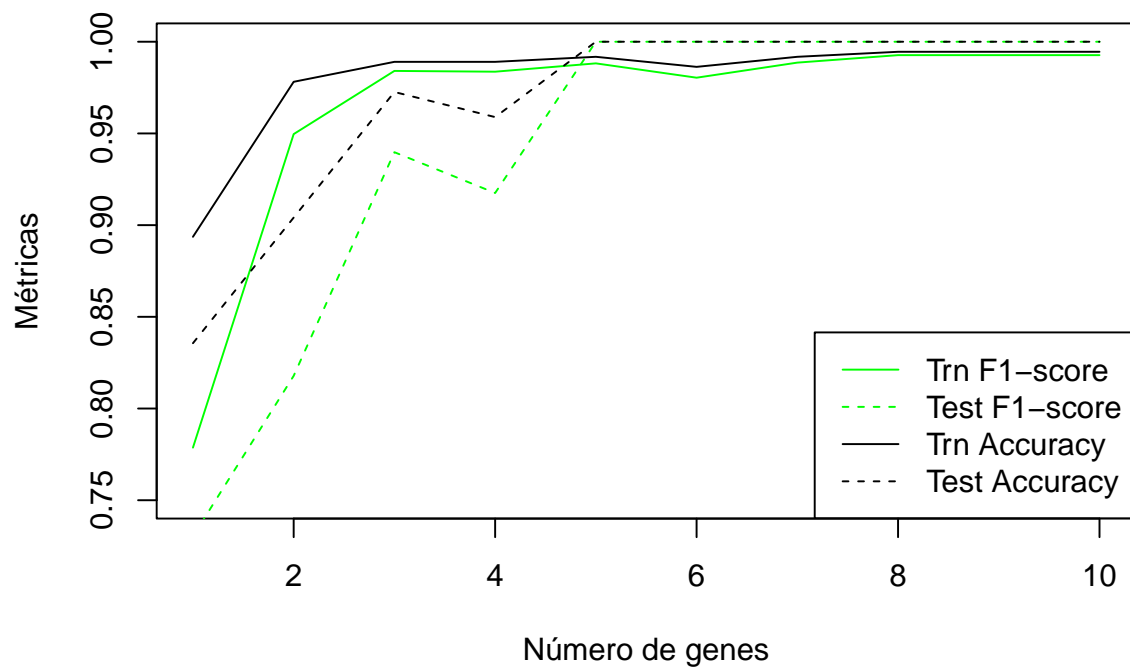
Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- rf_results_mrmr_test[1, 1:10]
Test_F1 <- rf_results_mrmr_test[2, 1:10]
```

```

Trn_Acc <- rf_results_mrmr_test[3,1:10]
Test_Acc <- rf_results_mrmr_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(rf_test_mrmr$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	4	0	0
	Healthy	0	9	1
	LIHC	1	0	58
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 97.26	F1 93.977	Sensitivity 96.102	Specificity 97.136
--------------------------	---------------------	------------------------------	------------------------------

Observando la primera gráfica podemos decir que el mejor número de genes para el algoritmo de clasificación Random Forest con el algoritmo de selección de características mRMR, es **3**, ya que obtiene muy buenas métricas y tiene bastante explicabilidad y presentan picos en ambas métricas.

3.2.2 Algoritmo de selección de características: RF

En esta sección se utilizará el método de selección de características Random Forest (RF). Se seguirán los mismos pasos aplicados anteriormente con mRMR para analizar los resultados obtenidos tanto en el entrenamiento como en el test, y así comparar su rendimiento.

En primer lugar, realizamos el entrenamiento del modelo Random Forest utilizando las 10 características seleccionadas por el método RF:

```
rf_trn_rf <- rf_trn(MLMatrix, MLLabels, vars_selected=FSRankingRF[1:10], numFold = 5)
save(rf_trn_rf, file = "rf_trn_rf")
```

```
load(file = "rf_trn_rf")
```

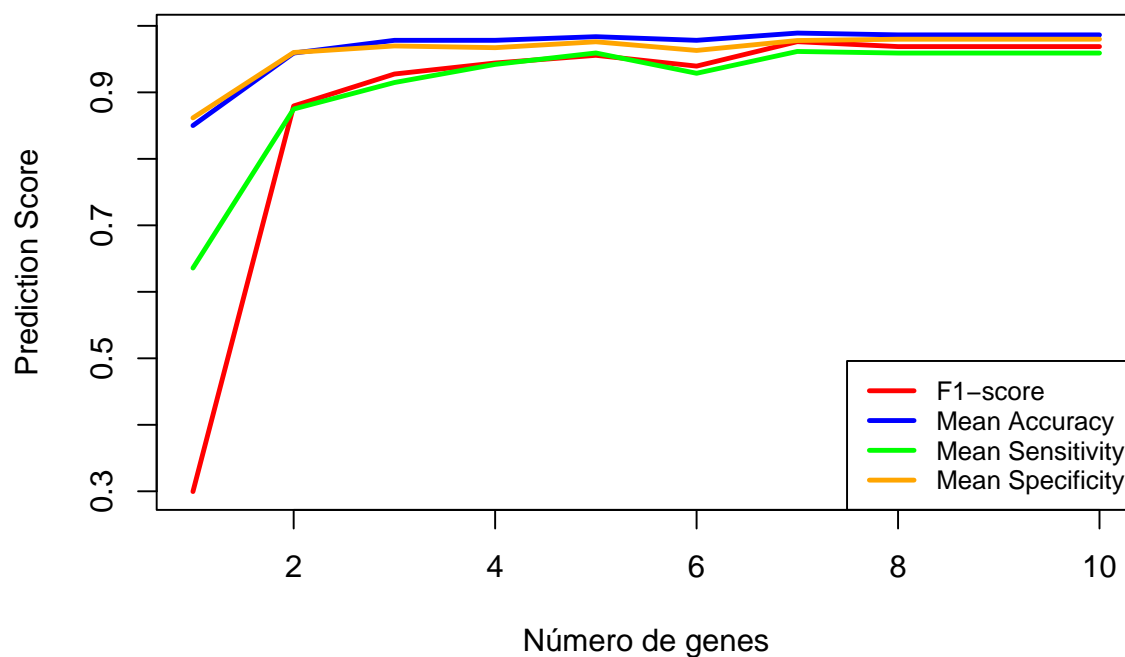
Se combinan las métricas principales (F1-score, Accuracy, Sensitivity y Specificity) para los 10 primeros genes seleccionados por RF en una sola matriz (`rf_results_rf_trn`), lo que facilita su visualización.

```
rf_results_rf_trn <- rbind(rf_trn_rf$F1Info$meanF1[1:10],
  rf_trn_rf$accuracyInfo$meanAccuracy, rf_trn_rf$sensitivityInfo$meanSensitivity,
  rf_trn_rf$specificityInfo$meanSpecificity)
save(rf_results_rf_trn, file = "rf_results_rf_trn")
```

```
load(file = "rf_results_rf_trn")
```

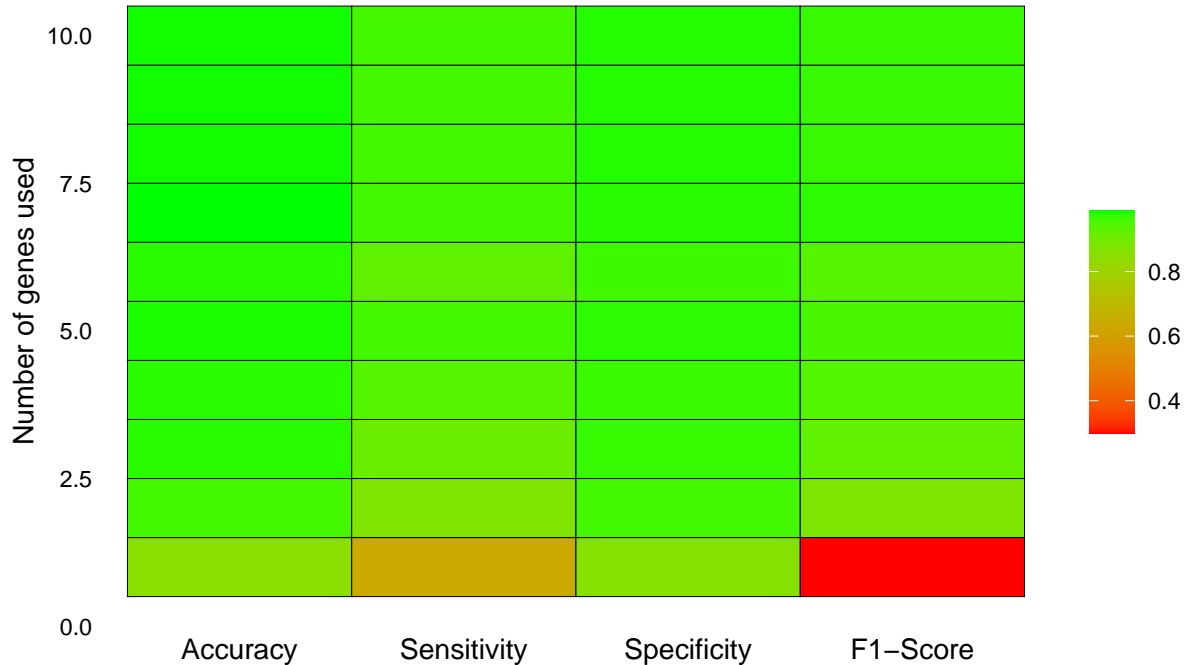
A continuación, se crea una gráfica que muestre el rendimiento de cada métrica en función del número de genes.

```
dataPlot(rf_results_rf_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Se mostrará un mapa de calor con los resultados globales del entrenamiento:

```
dataPlot(rf_trn_rf, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por RF y los best params obtenidos durante el entrenamiento:

```
rf_test_rf <- rf_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
  FSRankingRF[1:10], bestParameters=rf_trn_rf$bestParameters)
save(rf_test_rf, file = "rf_test_rf")
```

```
load(file = "rf_test_rf")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
rf_results_rf_test <- rbind(rf_trn_rf$F1Info$meanF1[1:10], rf_test_rf$f1Vector,
  rf_trn_rf$accuracyInfo$meanAccuracy, rf_test_rf$accVector)
save(rf_results_rf_test, file = "rf_results_rf_test")
```

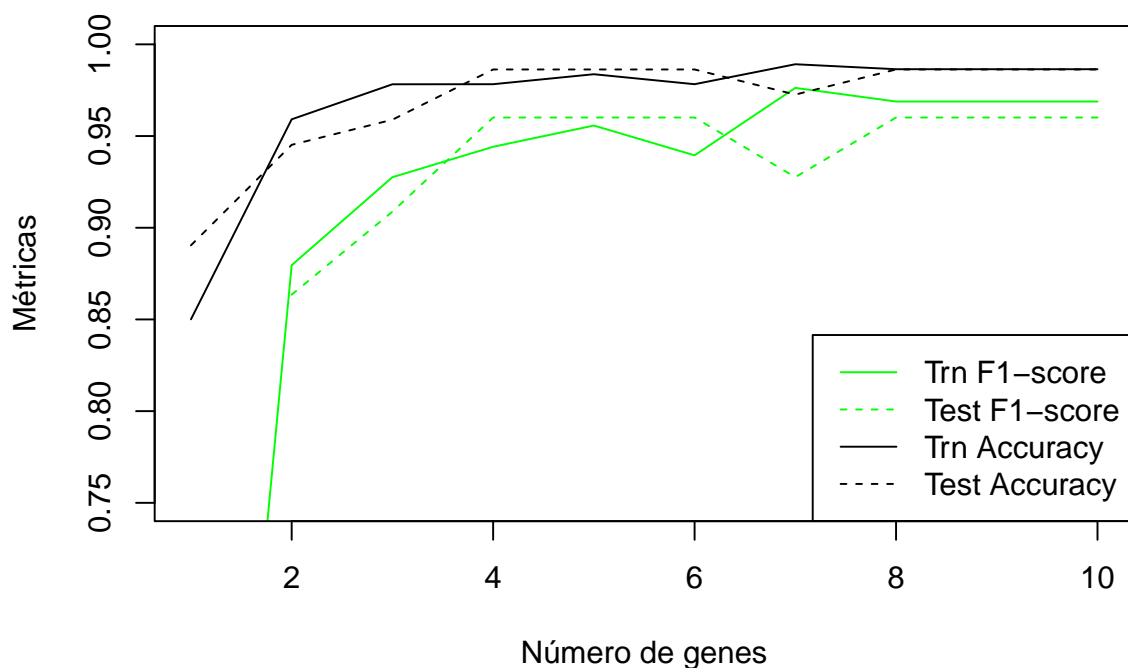
```
load(file = "rf_results_rf_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- rf_results_rf_test[1,1:10]
Test_F1 <- rf_results_rf_test[2,1:10]
Trn_Acc <- rf_results_rf_test[3,1:10]
Test_Acc <- rf_results_rf_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Por último, mostramos la matriz de confusión:

```

dataPlot(rf_test_rf$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	4	0	0
	Healthy	0	10	0
	LIHC	2	1	56
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
95.89	90.876	98.305	98.505

Este algoritmo de selección de características ha demostrado obtener métricas destacadas. Al analizar la gráfica, considero que la mejor opción es seleccionar **5 genes**, ya que alcanzan puntuaciones elevadas, lo que garantiza un buen rendimiento y proporciona una interpretación más clara y explicable de los resultados. También se podrían haber elegido 2 genes o 7 genes ya que también presentan buenas puntuaciones. Sin embargo, he decidido que utilizar **5 genes** es la mejor opción para no tener en exceso ni en defecto.

3.2.3 Algoritmo de selección de características: DA

Esta será una de las últimas prueba que se hará para el algoritmo Random Forest, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo Random Forest utilizando las 10 características seleccionadas por el método DA:

```
rf_trn_da <- rf_trn(MLMatrix, MLLabels, vars_selected=names(FSRankingDA[1:10]),
                    numFold = 5)
save(rf_trn_da, file = "rf_trn_da")

load(file = "rf_trn_da")
```

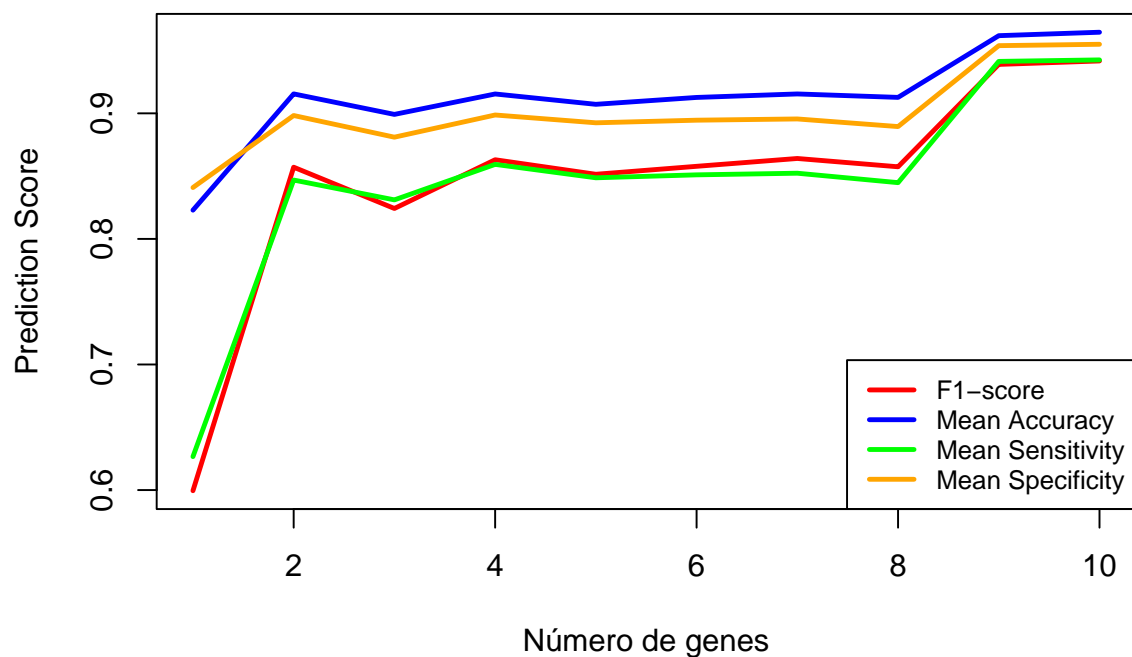
Calculamos las métricas de evaluación **F1-score**, **Accuracy**, **Sensitivity** y **Specificity** y almacenamos los resultados:

```
rf_results_da_trn<-rbind(rf_trn_da$F1Info$meanF1[1:10],
  rf_trn_da$accuracyInfo$meanAccuracy, rf_trn_da$sensitivityInfo$meanSensitivity,
  rf_trn_da$specificityInfo$meanSpecificity)
save(rf_results_da_trn,file = "rf_results_da_trn")
```

```
load(file = "rf_results_da_trn")
```

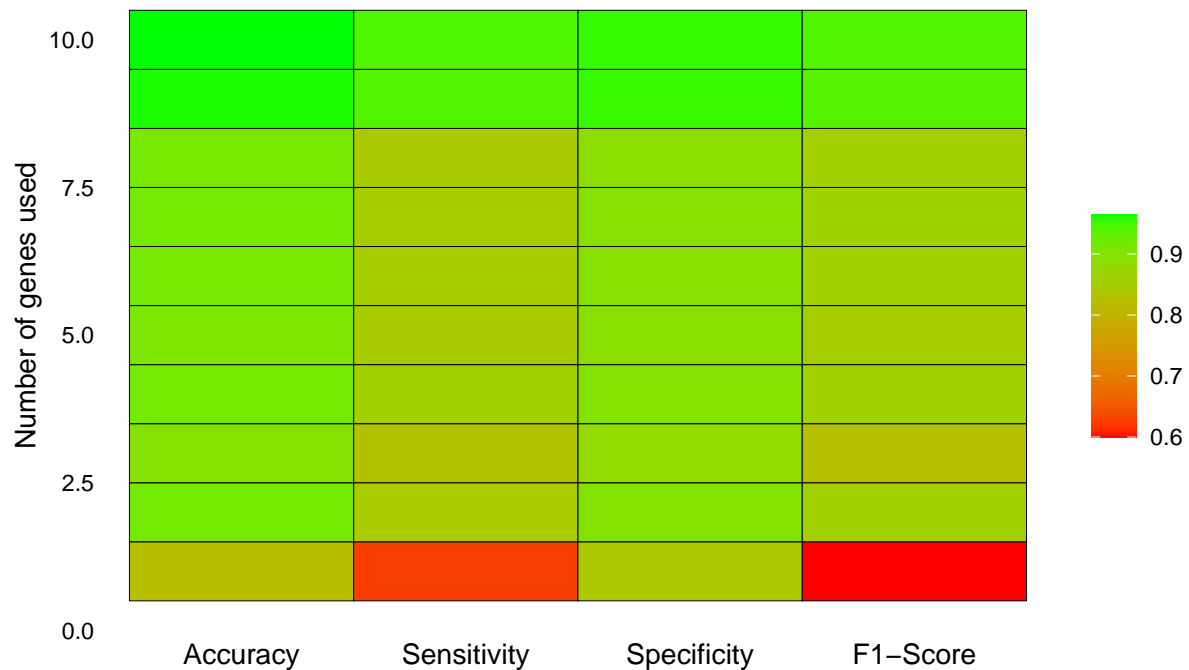
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(rf_results_da_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados:

```
dataPlot(rf_trn_da, MLLabels, mode = "heatmapResults")
```

A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por DA y los best params obtenidos durante el entrenamiento:

```
rf_test_da <- rf_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
  names(FSRankingDA[1:10]), bestParameters=rf_trn_da$bestParameters)
save(rf_test_da,file = "rf_test_da")
```

```
load(file = "rf_test_da")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
rf_results_da_test <- rbind(rf_trn_da$F1Info$meanF1[1:10], rf_test_da$f1Vector,
  rf_trn_da$accuracyInfo$meanAccuracy, rf_test_da$accVector)
save(rf_results_da_test,file = "rf_results_da_test")
```

```
load(file = "rf_results_da_test")
```

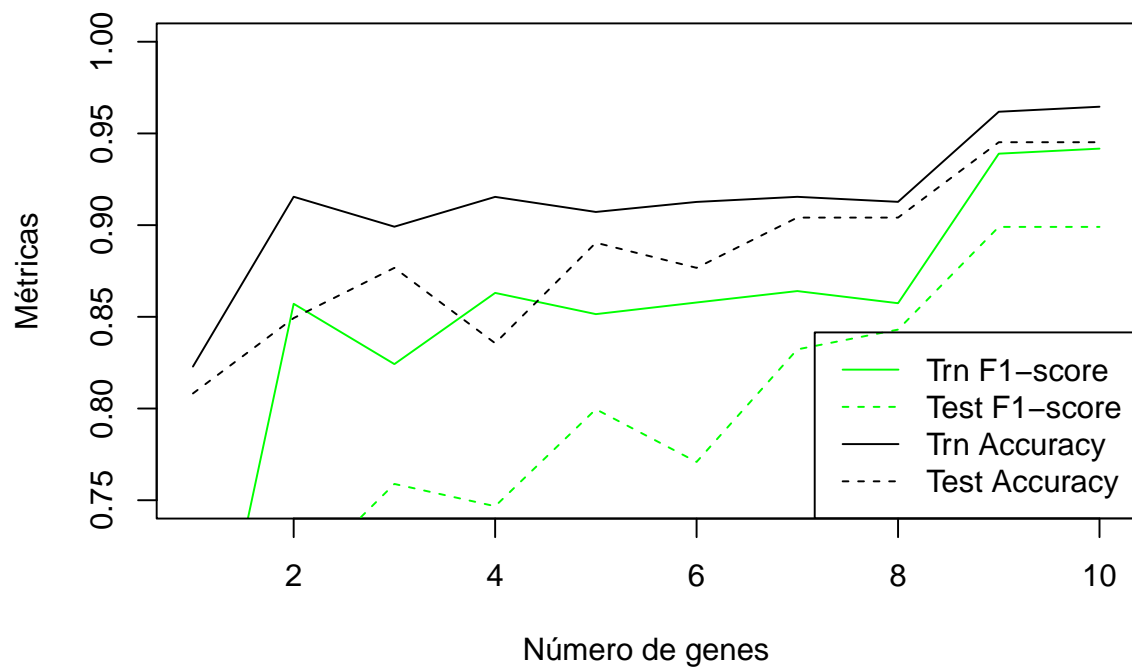
Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- rf_results_da_test[1,1:10]
Test_F1 <- rf_results_da_test[2,1:10]
```

```

Trn_Acc <- rf_results_da_test[3,1:10]
Test_Acc <- rf_results_da_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(rf_test_da$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	4	0	0
	Healthy	0	5	5
	LIHC	3	1	55
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 87.671	F1 75.888	Sensitivity 81.073	Specificity 86.117
---------------------------	---------------------	------------------------------	------------------------------

Este algoritmo de selección de características ha mostrado resultados inferiores en comparación con el anterior. Sin embargo, al analizar la gráfica, considero que **9 genes** es la mejor opción, ya que permite alcanzar aproximadamente un 95% de prediction score y ofrece una buena interpretabilidad del modelo y se obtiene picos en tanto en TRN como TEST en las métricas que utilizando dos genes. Otra opción sería elegir 4 genes si queremos tener un menor número de los mismo y también obtiene buenas métricas.

3.2.4 Algoritmo de selección de características: Lasso

Esta será la última prueba que se hará para el algoritmo Random Forest, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo Random Forest utilizando las 10 características seleccionadas por el método **Lasso**:

```
rf_trn_lasso <- rf_trn(MLMatrix, MLLabels, vars_selected = FSRankingLasso[1:10])
save(rf_trn_lasso, file = "rf_trn_lasso")
```

```
load(file = "rf_trn_lasso")
```

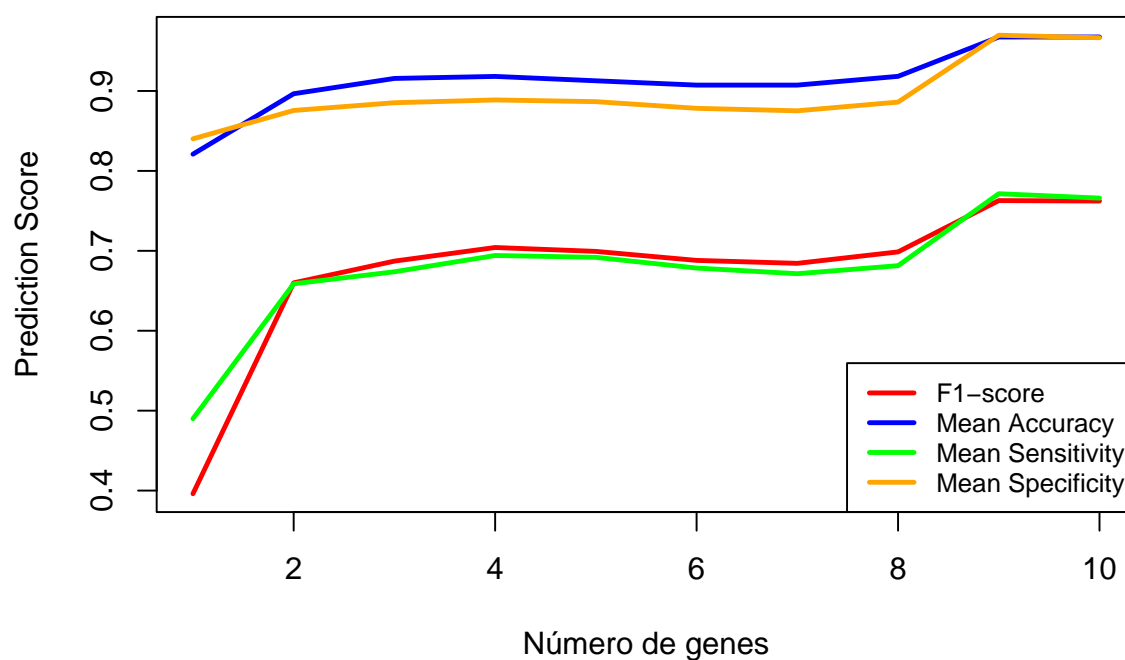
Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

```
rf_results_lasso_trn <- rbind(rf_trn_lasso$F1Info$meanF1[1:10],
  rf_trn_lasso$accuracyInfo$meanAccuracy, rf_trn_lasso$sensitivityInfo$meanSensitivity,
  rf_trn_lasso$specificityInfo$meanSpecificity)
save(rf_results_lasso_trn, file = "rf_results_lasso_trn")
```

```
load(file = "rf_results_lasso_trn")
```

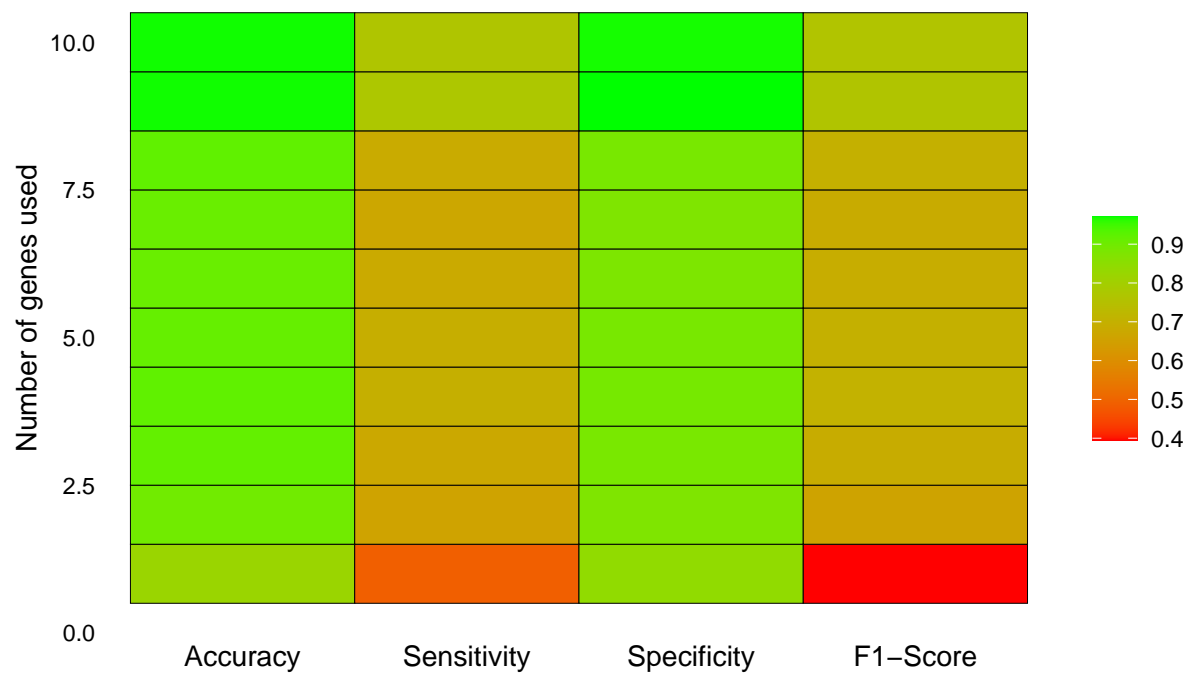
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(rf_results_lasso_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```

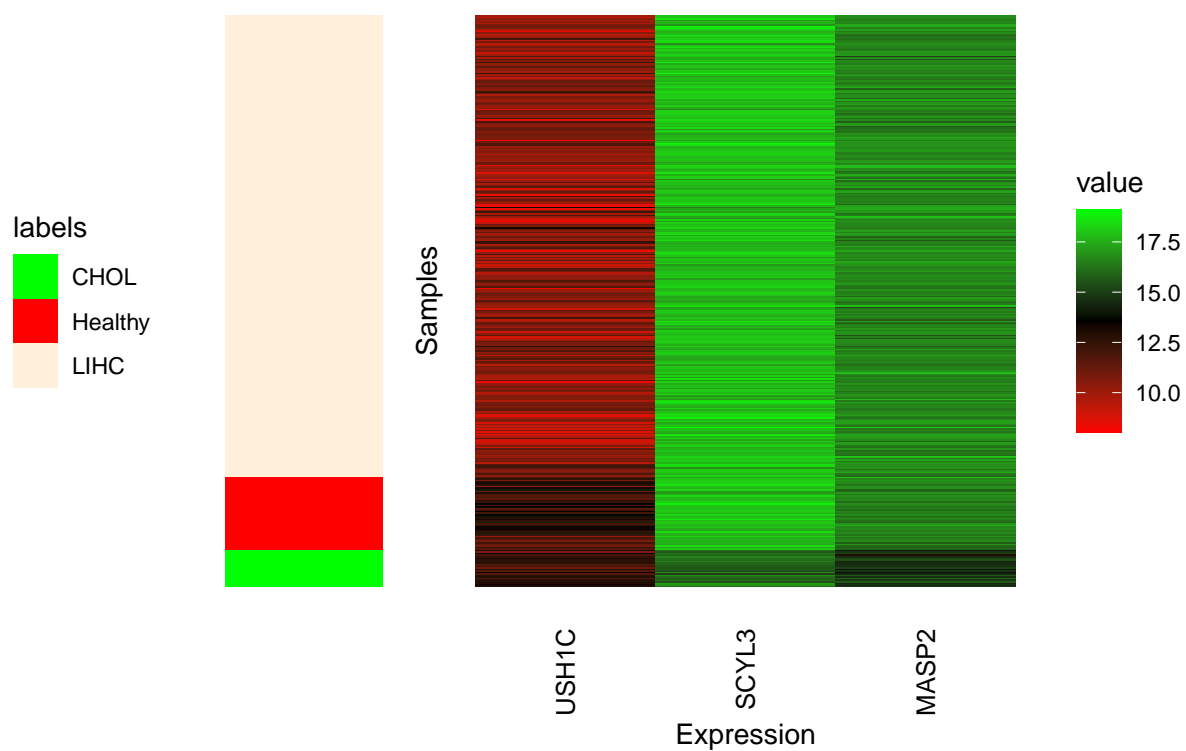


Mostramos un mapa de calor de los resultados y los valores correspondientes a las tres características principales seleccionadas:

```
dataPlot(rf_trn_lasso, MLLabels, mode = "heatmapResults")
```



```
dataPlot(t(MLMatrix[,FSRankingLasso[1:3]]), MLLabels, mode = "heatmap")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por Lasso y el valor de k obtenido durante el entrenamiento:

```
rf_test_lasso <- rf_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
                        FSRankingLasso[1:10], bestParameters=rf_trn_lasso$bestParameters)
save(rf_test_lasso,file = "rf_test_lasso")
```

```
load(file = "rf_test_lasso")
```

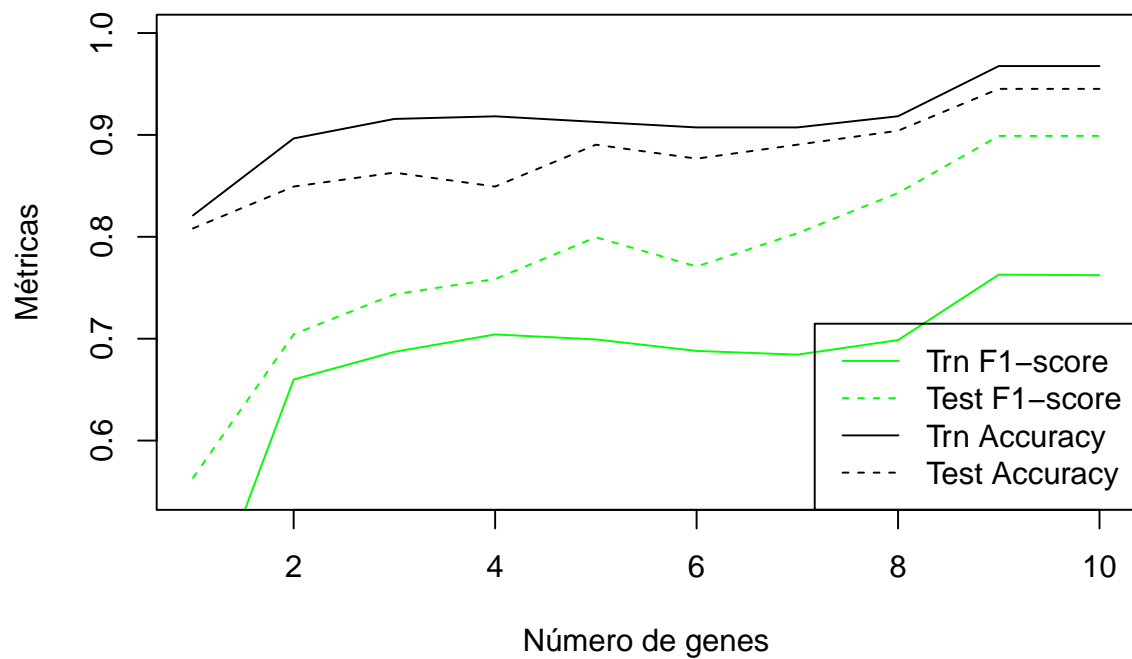
Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
rf_results_lasso_test <- rbind(rf_trn_lasso$F1Info$meanF1[1:10],rf_test_lasso$f1Vector,
                               rf_trn_lasso$accuracyInfo$meanAccuracy, rf_test_lasso$accVector)
save(rf_results_lasso_test,file = "rf_results_lasso_test" )
```

```
load(file = "rf_results_lasso_test")
```

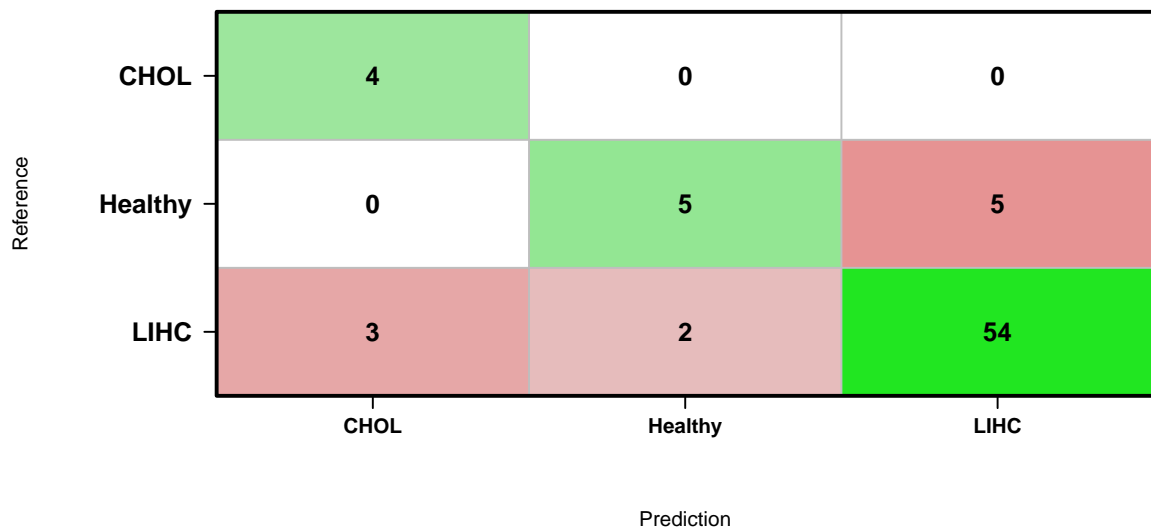
Visualizamos los resultados comparando train y test:

```
num_genes <- 1:10
Trn_F1 <- rf_results_lasso_test[1,1:10]
Test_F1 <- rf_results_lasso_test[2,1:10]
Trn_Acc <- rf_results_lasso_test[3,1:10]
Test_Acc <- rf_results_lasso_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.55,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )
```



Mostramos la matriz de confusión en el conjunto de test:

```
dataPlot(rf_test_lasso$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")
```



DETAILS

Accuracy	F1	Sensitivity	Specificity
86.301	74.359	80.508	85.588

En esta gráfica podemos observar varios picos, dependiendo del número de genes utilizados. En mi opinión, la opción más adecuada sería emplear nueve genes, ya que parece ofrecer una buena explicabilidad con ese número. Aunque las métricas de TRN también parecen tener un pico con dos genes. Sin embargo, creo que elegiré **9 genes** ya que se muestra un pico superior en ambas métricas.

3.2.5 Conclusiones

De nuevo, para el algoritmo **Random Forest**, tras analizar los resultados obtenidos con diferentes métodos de selección de características, considero que la mejor opción es utilizar **mRMR con un número de 3 genes seleccionados**. Con esta configuración, se logra superar el 90% de predicción, alcanzando además los valores más altos de **Accuracy** y **F1-score** en comparación con los demás algoritmos de selección de características.

3.3 Algoritmo de clasificación: SVM

En este apartado se probarán distintos algoritmos de selección de características para el algoritmo, como se ha comentado anteriormente son cuatro.

3.3.1 Algoritmo de selección de características: mRMR

En primer lugar, realizamos el entrenamiento del modelo SVM utilizando las 10 características seleccionadas por el método mRMR:

```
svm_trn_mrmr <- svm_trn(MLMatrix, MLLabels,vars_selected=FSRankingmRMR[1:10],
                        numFold = 5)
save(svm_trn_mrmr,file = "svm_trn_mrmr")
```

```
load(file = "svm_trn_mrmr")
```

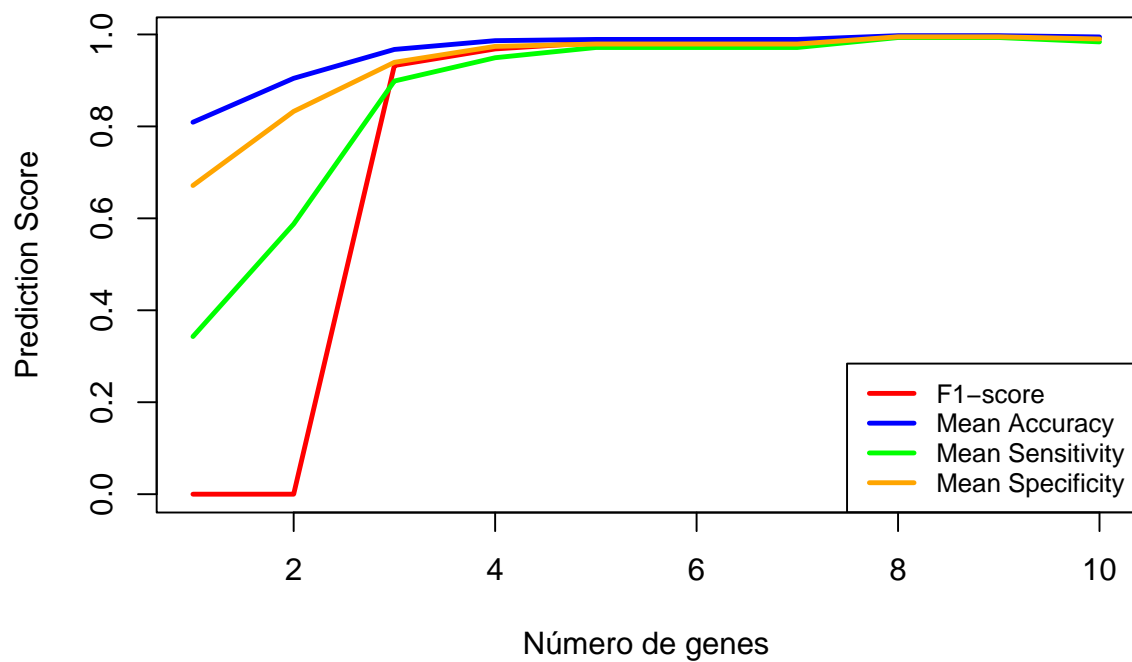
Calculamos las métricas de evaluación **F1-score**, **Accuracy**, **Sensitivity** y **Specificity** y almacenamos los resultados:

```
svm_results_mrmr_trn <- rbind(svm_trn_mrmr$F1Info$meanF1[1:10],
                              svm_trn_mrmr$accuracyInfo$meanAccuracy, svm_trn_mrmr$sensitivityInfo$meanSensitivity,
                              svm_trn_mrmr$specificityInfo$meanSpecificity)
save(svm_results_mrmr_trn, file = "svm_results_mrmr_trn")
```

```
load(file = "svm_results_mrmr_trn")
```

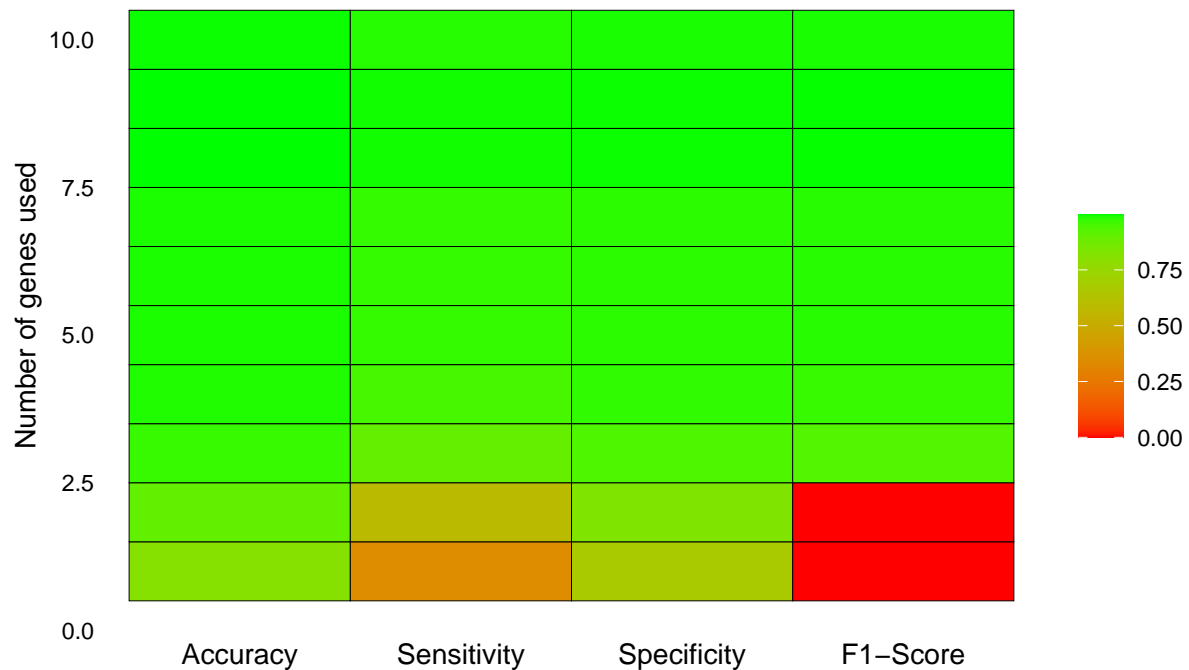
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:


```
dataPlot(svm_results_mrmr_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
"Mean Sensitivity", "Mean Specificity"), mode = "classResults",
xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados:

```
dataPlot(svm_trn_mrmr, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por mRMR y los `best params` obtenidos durante el entrenamiento.

Quitamos los guiones para que no de error y calculamos el `svm_test`:

```
vars_selected <- make.names(names(FSRankingmRMR))
svm_test_mrmr <- svm_test(MLMatrix,MLLabels,XTest,YTest,vars_selected=
  vars_selected[1:10], bestParameters = svm_trn_mrmr$bestParameters)
save(svm_test_mrmr, file = "svm_test_mrmr")
```

```
load(file = "svm_test_mrmr" )
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
svm_results_mrmr_test <- rbind(svm_trn_mrmr$F1Info$meanF1[1:10],svm_test_mrmr$f1Vector,
  svm_trn_mrmr$accuracyInfo$meanAccuracy, svm_test_mrmr$accVector)
save(svm_results_mrmr_test,file = "svm_results_mrmr_test")
```

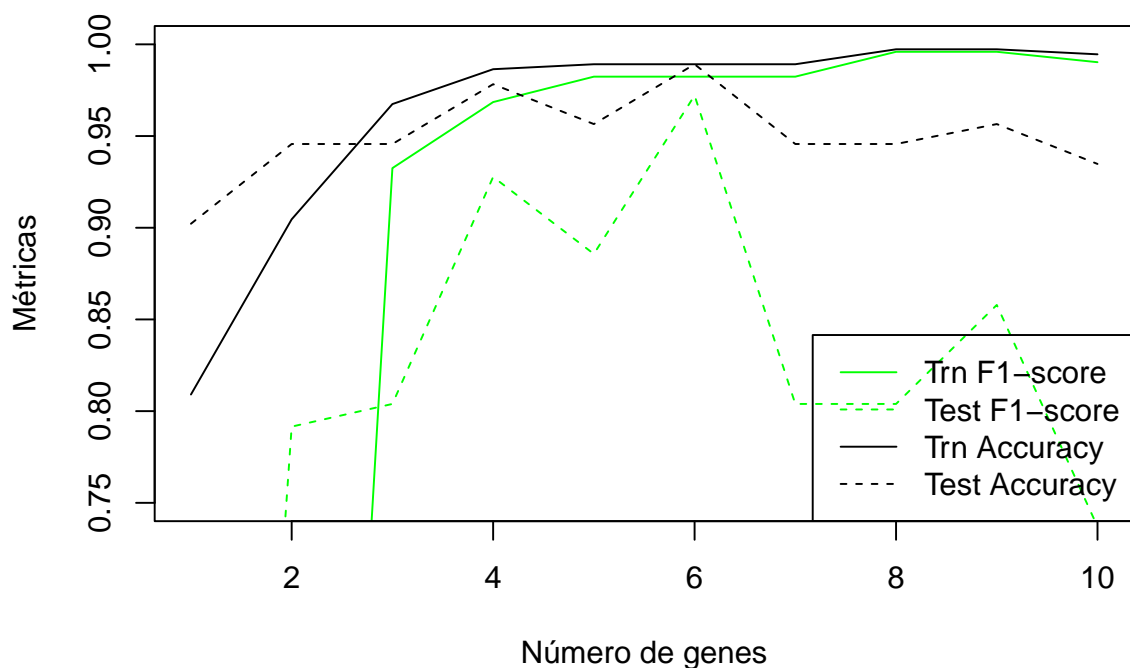
```
load(file = "svm_results_mrmr_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- svm_results_mrmr_test[1,1:10]
Test_F1 <- svm_results_mrmr_test[2,1:10]
Trn_Acc <- svm_results_mrmr_test[3,1:10]
Test_Acc <- svm_results_mrmr_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(svm_test_mrmr$cfMats[[2]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	0	0	4
	Healthy	0	9	1
	LIHC	0	1	58
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
91.781	61.694	62.768	87.566

Este algoritmo de clasificación SVM presenta picos en la primera gráfica como podemos observar por lo que el mejor número de genes a elegir sería **5**, ya que aproximadamente dan una explicabilidad del 96% aproximadamente.

3.3.2 Algoritmo de selección de características: RF

En esta sección se utilizará el método de selección de características Random Forest (RF). Se seguirán los mismos pasos aplicados anteriormente con mRMR para analizar los resultados obtenidos tanto en el entrenamiento como en el test, y así comparar su rendimiento.

En primer lugar, realizamos el entrenamiento del modelo SVM utilizando las 10 características seleccionadas por el método RF:

```
svm_trn_rf <- svm_trn(MLMatrix,MLLabels,vars_selected=FSRankingRF[1:10],numFold=5)
save(svm_trn_rf,file = "svm_trn_rf")
```

```
load(file = "svm_trn_rf")
```

Se combinan las métricas principales (F1-score, Accuracy, Sensitivity y Specificity) para los 10 primeros genes seleccionados por RF en una sola matriz, lo que facilita su visualización.

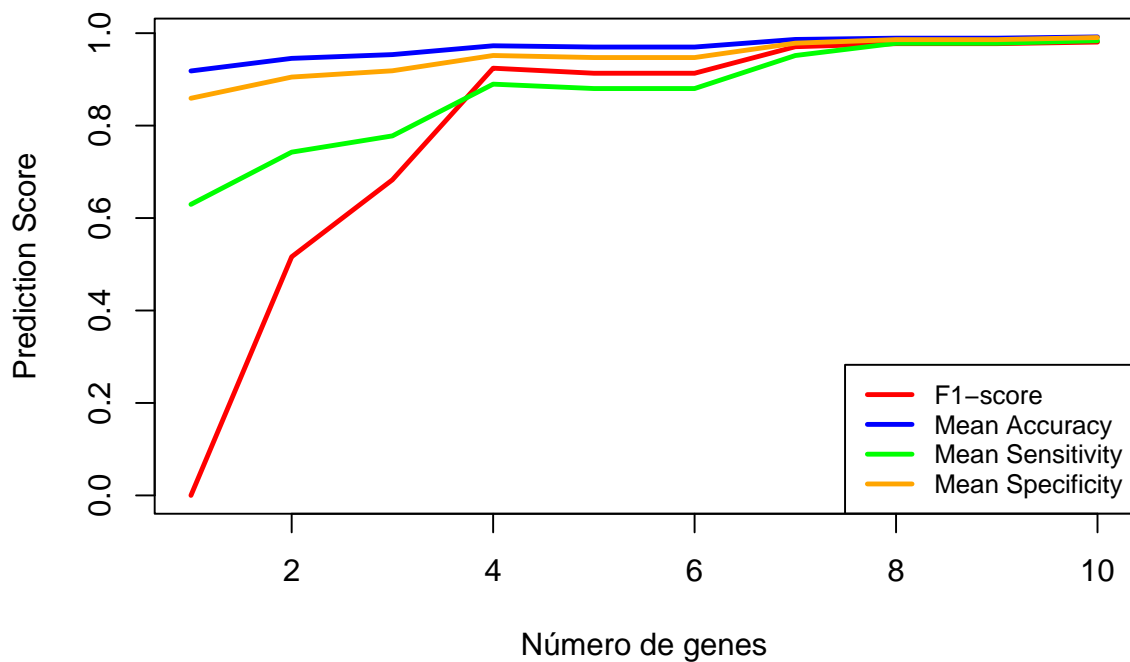
```
svm_results_rf_trn <- rbind(svm_trn_rf$F1Info$meanF1[1:10],
  svm_trn_rf$accuracyInfo$meanAccuracy, svm_trn_rf$sensitivityInfo$meanSensitivity,
  svm_trn_rf$specificityInfo$meanSpecificity)
```

```
save(svm_results_rf_trn,file = "svm_results_rf_trn")
```

```
load(file = "svm_results_rf_trn")
```

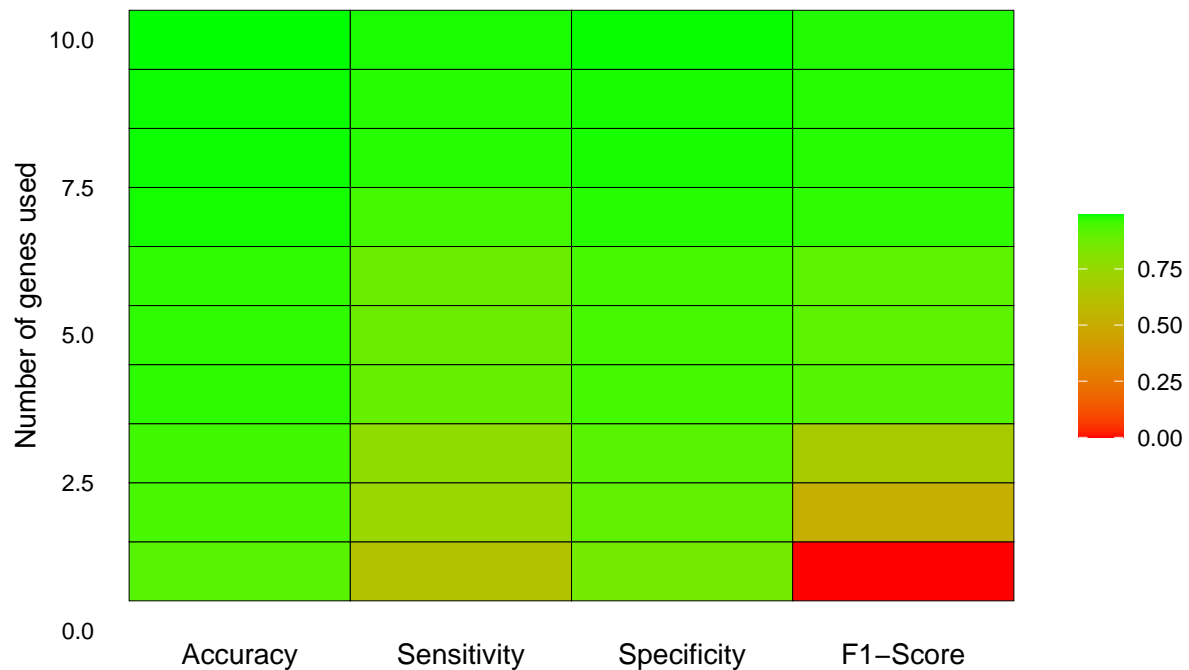
A continuación, se crea una gráfica que muestre el rendimiento de cada métrica en función del número de genes.

```
dataPlot(svm_results_rf_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Se mostrará un mapa de calor con los resultados globales del entrenamiento:

```
dataPlot(svm_trn_rf, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por RF y los best params obtenidos durante el entrenamiento:

```
svm_test_rf <- svm_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
                        FSRankingRF[1:10],bestParameters=svm_trn_rf$bestParameters)

save(svm_test_rf,file = "svm_test_rf")
```

```
load(file = "svm_test_rf")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
svm_results_rf_test <- rbind(svm_trn_rf$F1Info$meanF1[1:10], svm_test_rf$f1Vector,
                             svm_trn_rf$accuracyInfo$meanAccuracy, svm_test_rf$accVector)

save(svm_results_rf_test,file = "svm_results_rf_test")
```

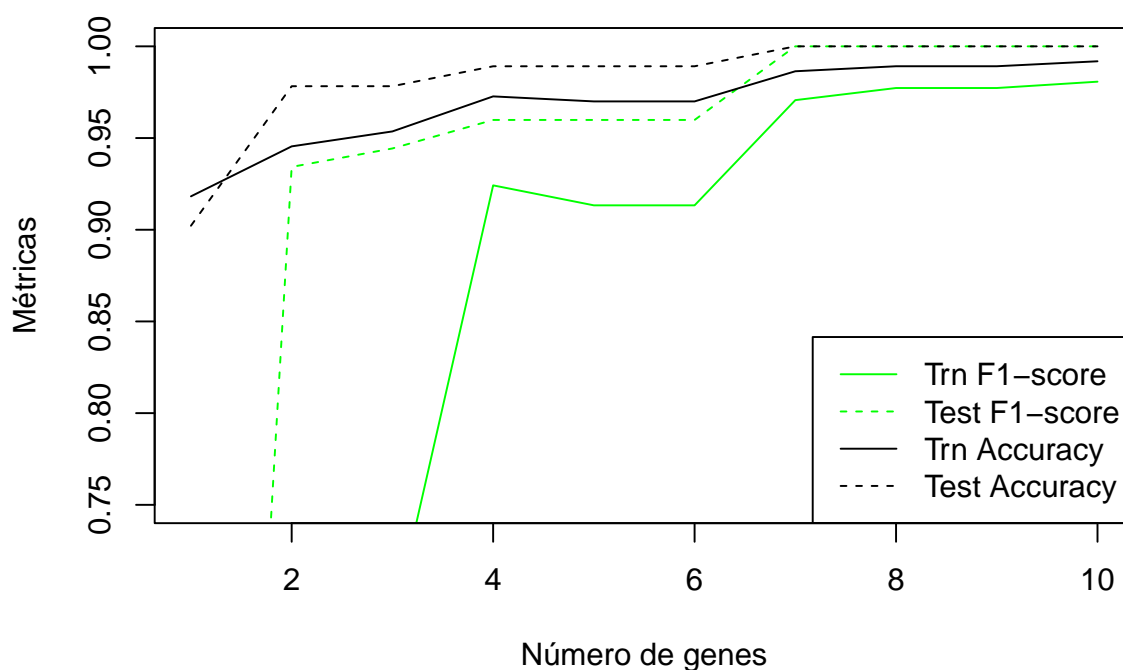
```
load(file = "svm_results_rf_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- svm_results_rf_test[1,1:10]
Test_F1 <- svm_results_rf_test[2,1:10]
Trn_Acc <- svm_results_rf_test[3,1:10]
Test_Acc <- svm_results_rf_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Por último, se mostrará la matriz de confusión:

```

dataPlot(svm_test_rf$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	6	1	0
	Healthy	0	11	0
	LIHC	0	1	73
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 97.826	F1 94.431	Sensitivity 94.788	Specificity 99.177
---------------------------	---------------------	------------------------------	------------------------------

En la gráfica se aprecia un pico en el número de genes igual a 4, lo que indica una alta capacidad explicativa con solo cuatro genes, pero si queremos más precisión lo suyo sería elegir el numero de genes igual a 8 ya que estos superan el 95%. Por tanto, el número de genes elegidos será 4 ya que con este número se supera el 90%.

3.3.3 Algoritmo de selección de características: DA

Esta será una de las últimas pruebas que se hará para el algoritmo SVM, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo SVM utilizando las 10 características seleccionadas por el método DA:

```
svm_trn_da <- svm_trn(MLMatrix, MLLabels, vars_selected = names(FSRankingDA[1:10]),
                      numFold = 5)
```

```
save(svm_trn_da,file = "svm_trn_da")
```

```
load(file = "svm_trn_da")
```

Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

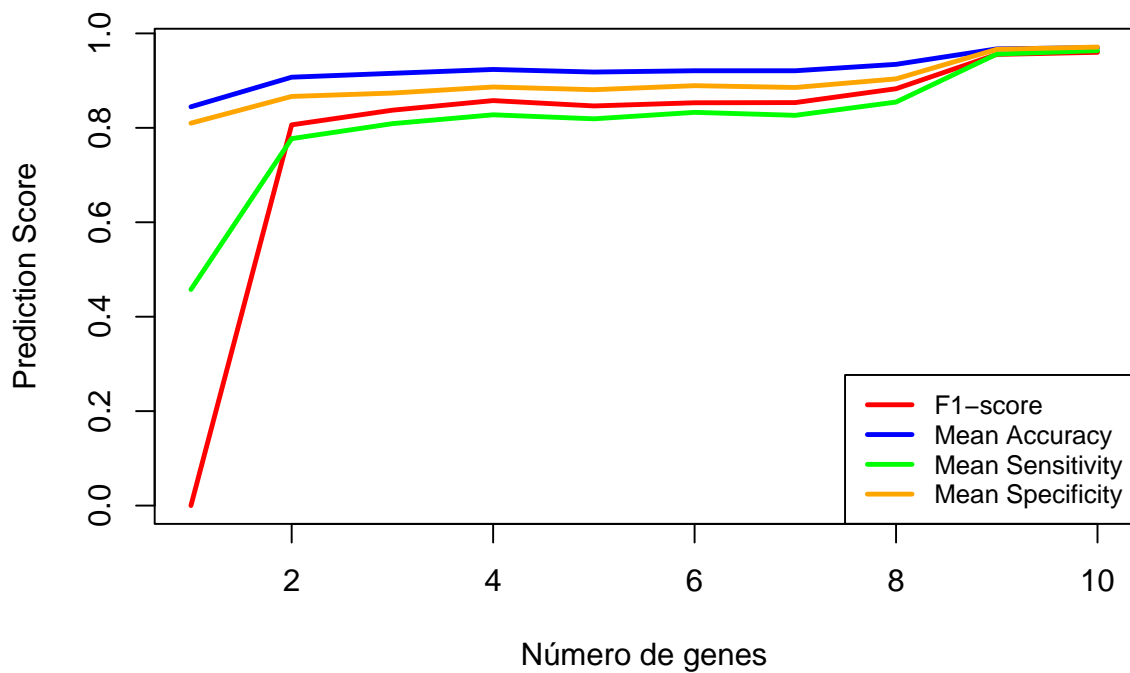

```
svm_results_da_trn <- rbind(svm_trn_da$F1Info$meanF1[1:10],
  svm_trn_da$accuracyInfo$meanAccuracy, svm_trn_da$sensitivityInfo$meanSensitivity,
  svm_trn_da$specificityInfo$meanSpecificity)
```

```
save(svm_results_da_trn,file = "svm_results_da_trn")
```

```
load(file = "svm_results_da_trn")
```

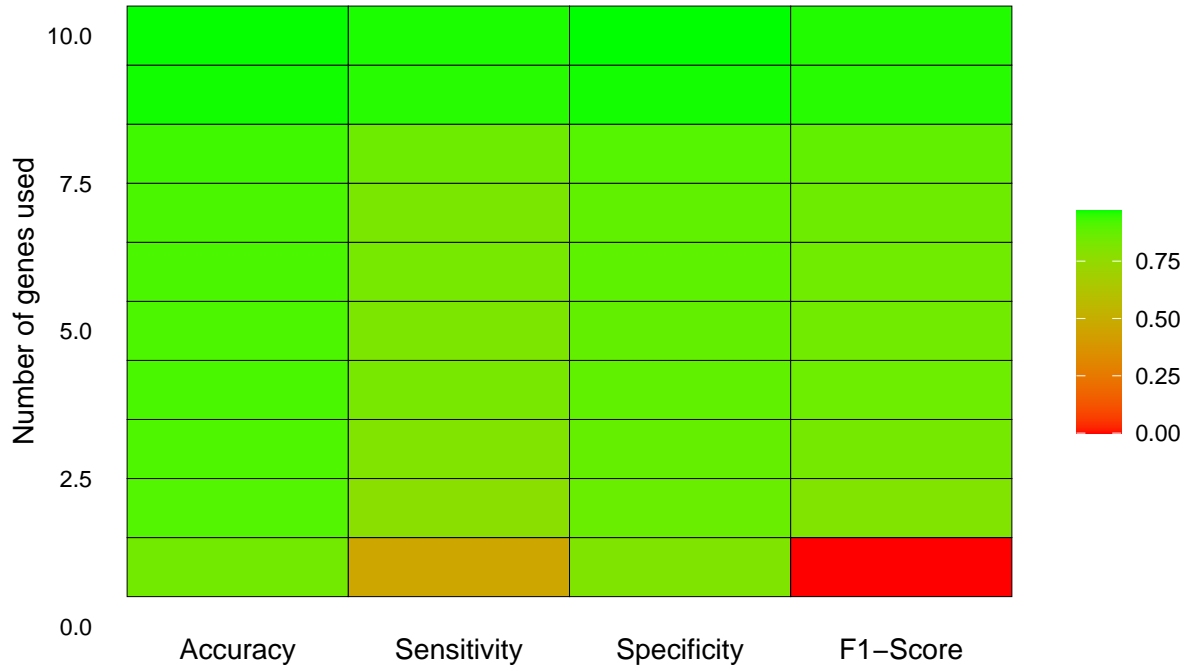
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(svm_results_da_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados:

```
dataPlot(svm_trn_da, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por DA y los best params obtenidos durante el entrenamiento:

```
svm_test_da <- svm_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
  names(FSRankingDA[1:10]), bestParameters=svm_trn_da$bestParameters)
save(svm_test_da,file = "svm_test_da")
```

```
load(file = "svm_test_da")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
svm_results_da_test <- rbind(svm_trn_da$F1Info$meanF1[1:10], svm_test_da$f1Vector,
  svm_trn_da$accuracyInfo$meanAccuracy, svm_test_da$accVector)
save(svm_results_da_test,file = "svm_results_da_test")
```

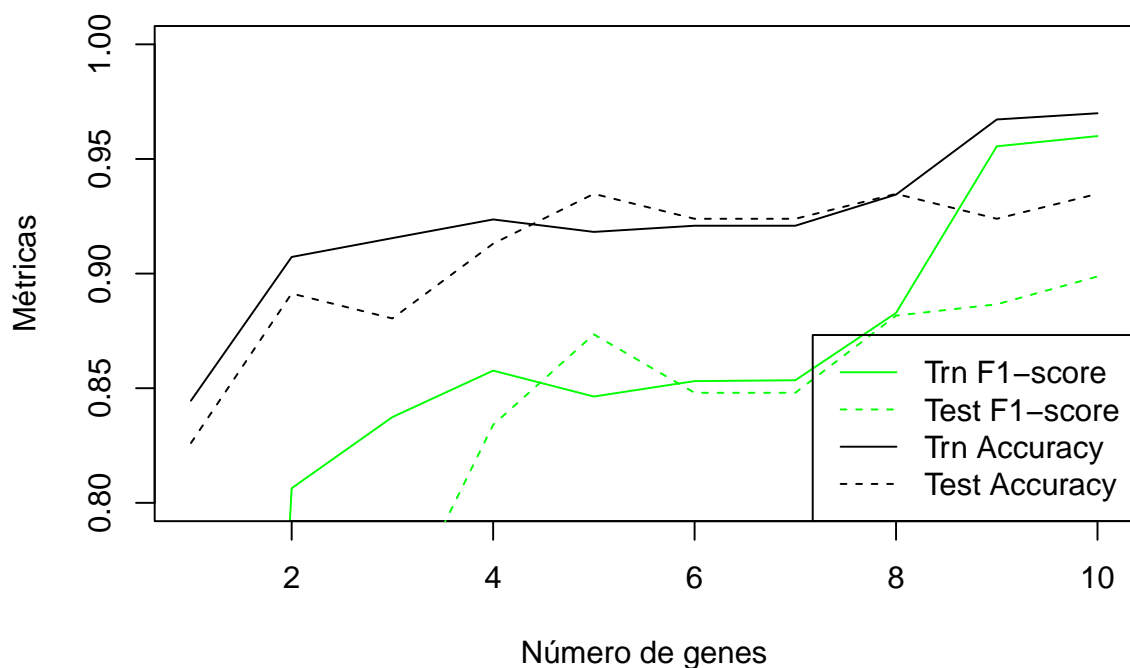
```
load(file = "svm_results_da_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- svm_results_da_test[1,1:10]
Test_F1 <- svm_results_da_test[2,1:10]
Trn_Acc <- svm_results_da_test[3,1:10]
Test_Acc <- svm_results_da_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.80,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(svm_test_da$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	6	0	1
	Healthy	1	4	6
	LIHC	0	3	71
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 88.043	F1 74.527	Sensitivity 72.675	Specificity 85.41
---------------------------	---------------------	------------------------------	-----------------------------

En este algoritmo de selección de características, se puede observar que el número de genes con mayor explicabilidad es 9, ya que alcanza casi un 95%. Sin embargo, son bastantes genes y con un número de genes igual a 4 se puede observar que hay picos en las métricas de TRN y presenta una explicabilidad de más del 85%. He decidido utilizar **4 genes**.

3.3.4 Algoritmo de selección de características: Lasso

Esta será la última prueba que se hará para el algoritmo SVM, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo SVM utilizando las 10 características seleccionadas por el método Lasso:

```
svm_trn_lasso <- svm_trn(MLMatrix, MLLabels, vars_selected =
                        FSRankingLasso[1:10], numFold = 5)
save(svm_trn_lasso, file = "svm_trn_lasso")
```

```
load(file = "svm_trn_lasso")
```

Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

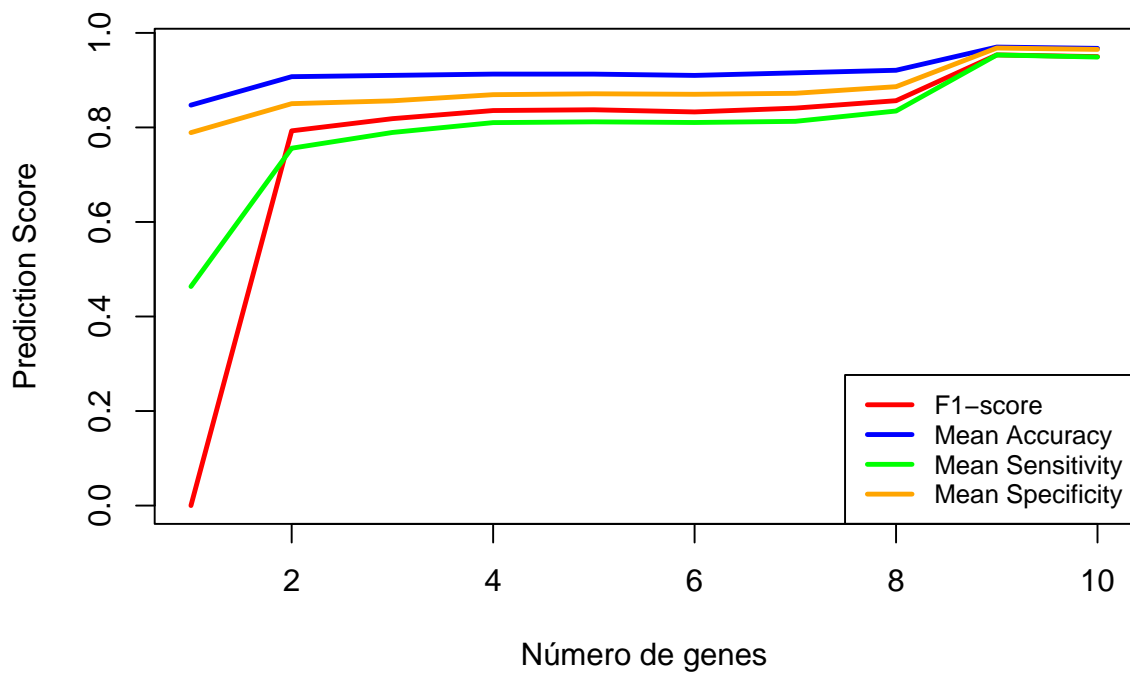
```
svm_results_lasso_trn <- rbind(svm_trn_lasso$F1Info$meanF1[1:10],
svm_trn_lasso$accuracyInfo$meanAccuracy, svm_trn_lasso$sensitivityInfo$meanSensitivity,
svm_trn_lasso$specificityInfo$meanSpecificity)
```

```
save(svm_results_lasso_trn,file = "svm_results_lasso_trn")
```

```
load(file = "svm_results_lasso_trn")
```

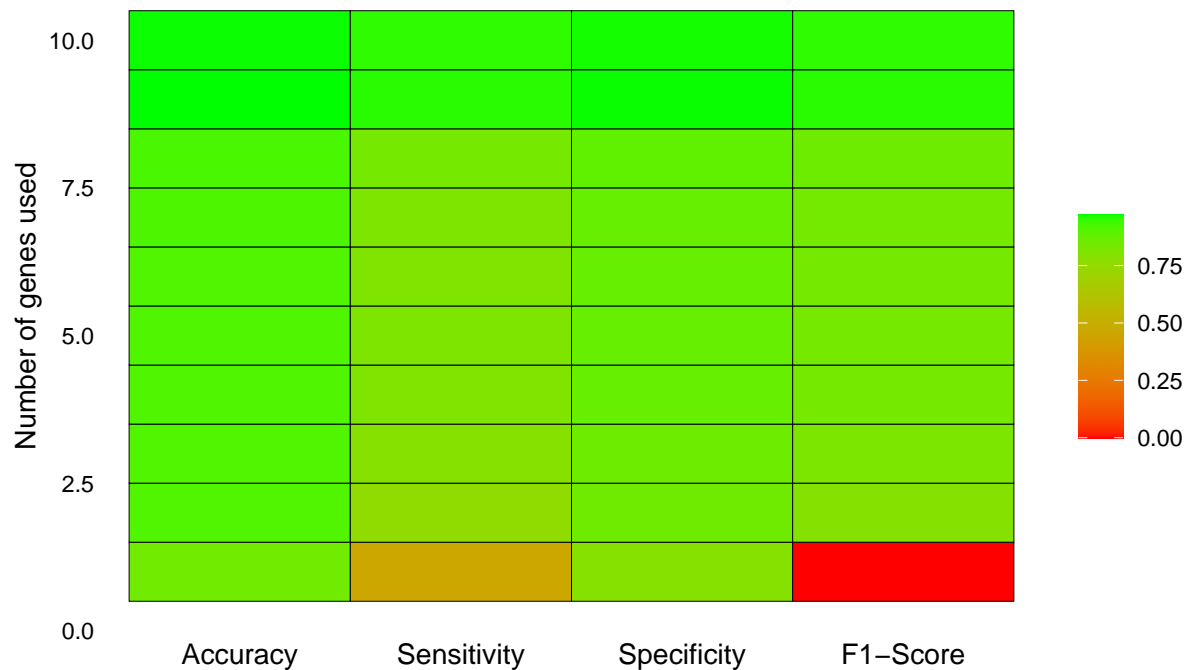
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(svm_results_lasso_trn, MLLabels, legend = c("F1-score","Mean Accuracy",
"Mean Sensitivity","Mean Specificity"), mode = "classResults",
xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados:

```
dataPlot(svm_trn_lasso, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por Lasso y los best params obtenidos durante el entrenamiento:

```
svm_test_lasso <- svm_test(MLMatrix, MLLabels, XTest, YTest, vars_selected=
  FSRankingLasso[1:10], bestParameters=svm_trn_lasso$bestParameters)
save(svm_test_lasso, file = "svm_test_lasso")
```

```
load(file = "svm_test_lasso")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
svm_results_lasso_test <- rbind(svm_trn_lasso$F1Info$meanF1[1:10],
  svm_test_lasso$f1Vector, svm_trn_lasso$accuracyInfo$meanAccuracy,
  svm_test_lasso$accVector)
save(svm_results_lasso_test, file = "svm_results_lasso_test")
```

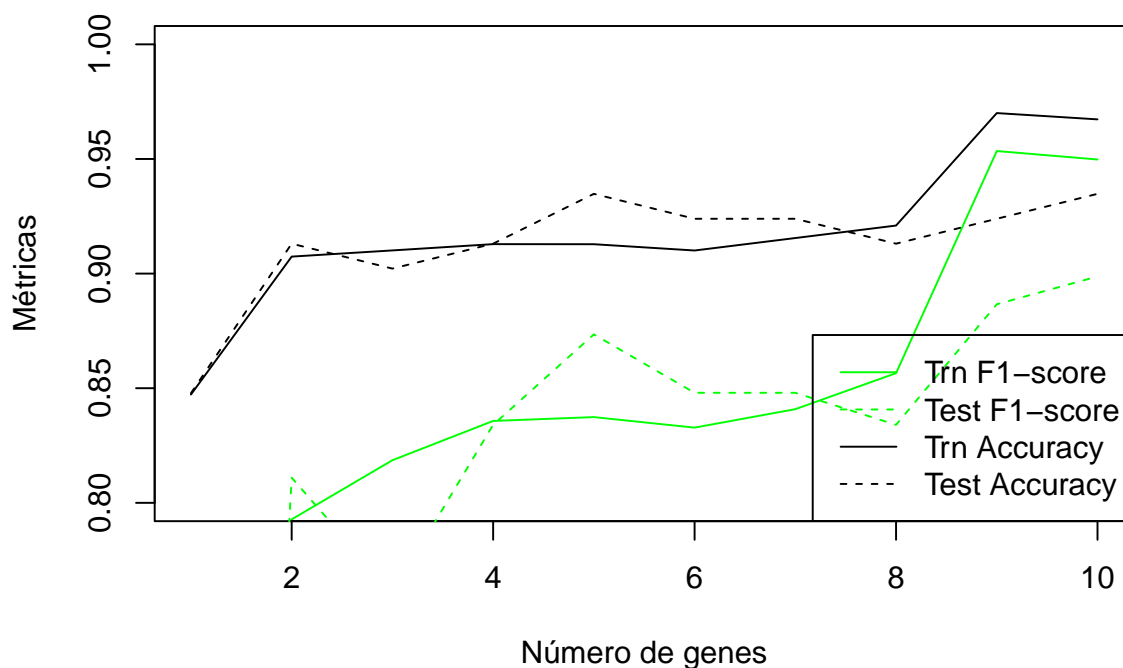
```
load(file = "svm_results_lasso_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- svm_results_lasso_test[1,1:10]
Test_F1 <- svm_results_lasso_test[2,1:10]
Trn_Acc <- svm_results_lasso_test[3,1:10]
Test_Acc <- svm_results_lasso_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.80,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(svm_test_lasso$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	6	1	0
	Healthy	1	4	6
	LIHC	0	1	73
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
90.217	76.066	73.576	87.674

En este algoritmo de selección de características, se puede observar que el número de genes con mayor explicabilidad es 9, ya que alcanza más de un 95%. Sin embargo, son bastantes genes y con un número de genes igual a 4 se puede observar que hay picos en las métricas de TRN y presenta una explicabilidad alrededor del 85% por lo que será el elegido.

3.3.5 Conclusiones

Al igual que en los dos anteriores modelos de clasificación, tras analizar los resultados obtenidos con diferentes métodos de selección de características, considero que la mejor opción es utilizar **Random Forest** con un número de **4 genes** seleccionados. Con esta configuración, se encuentra un 100% de predicción, alcanzando además los valores más altos de **Accuracy** y **F1-score** en comparación con los demás algoritmos de selección de características.

3.4 Algoritmo de clasificación: Redes neuronales

Este apartado consistirá en probar el algoritmo de clasificación de Redes Neuronales con distintos algoritmos de selección de características que han sido los mismos que se han visto anteriormente en los otros apartados.

3.4.1 Algoritmo de selección de características: mRMR

```
mn_trn <- function(data, labels, vars_selected, numFold = 10, hidden = c(5), maxit = 200) {  
  if (!is.data.frame(data) && !is.matrix(data)) {  
    stop("The data argument must be a dataframe or a matrix.")  
  }  
  if (dim(data)[1] != length(labels)) {  
    stop("The length of the rows of the argument data must be the same as the length  
      of the labels. Please ensure that the rows are the samples and the columns  
      are the variables.")  
  }  
  
  if (!is.character(labels) && !is.factor(labels)) {  
    stop("The class of the labels parameter must be a character vector or factor.")  
  }  
  if (is.character(labels)) {  
    labels <- as.factor(labels)  
  }  
  
  if (numFold %% 1 != 0 || numFold == 0) {  
    stop("The numFold argument must be an integer and greater than 0.")  
  }  
  
  data <- as.data.frame(apply(data, 2, as.double))  
  data <- data[, vars_selected]  
  
  data <- vapply(data, function(x) {  
    max <- max(x)  
    min <- min(x)  
    if (max > min) {  
      x <- ((x - min) / (max - min)) * 2 - 1  
    } else {  
      x  
    }  
  }, double(nrow(data)))  
  
  data <- as.data.frame(data)  
  
  fitControl <- trainControl(method = "cv", number = numFold)
```

```

cat("Training Neural Network with cross-validation...\n")

dataForTraining <- cbind(data, labels)
colnames(dataForTraining) <- make.names(colnames(dataForTraining))

nn_grid <- expand.grid(
  size = hidden, # Hidden layer sizes
  decay = c(0, 0.01, 0.1, 0.5) # Regularization parameter
)

nn_model <- train(
  labels ~ .,
  data = dataForTraining,
  method = "nnet",
  trControl = fitControl,
  tuneGrid = nn_grid,
  maxit = maxit,
  trace = FALSE,
  preProcess = c("center", "scale")
)

bestParameters <- nn_model$bestTune
cat(paste("Optimal size:", bestParameters$size, "\n"))
cat(paste("Optimal decay:", bestParameters$decay, "\n"))

acc_cv <- matrix(OL, nrow = numFold, ncol = dim(data)[2])
sens_cv <- matrix(OL, nrow = numFold, ncol = dim(data)[2])
spec_cv <- matrix(OL, nrow = numFold, ncol = dim(data)[2])
f1_cv <- matrix(OL, nrow = numFold, ncol = dim(data)[2])
cfMatList <- list()

lengthValFold <- dim(data)[1] / numFold
positions <- rep(seq_len(dim(data)[1]))
randomPositions <- sample(positions)
data <- data[randomPositions, ]
labels <- labels[randomPositions]

for (i in seq_len(numFold)) {

```

```

cat(paste("Training fold", i, "...\\n"))

valFold <- seq(round((i - 1) * lengthValFold + 1), round(i * lengthValFold))
trainDataCV <- setdiff(seq_len(dim(data)[1]), valFold)
testDataset <- data[valFold, ]
trainingDataset <- data[trainDataCV, ]
labelsTrain <- labels[trainDataCV]
labelsTest <- labels[valFold]

colNames <- colnames(trainingDataset)

for (j in seq_len(length(vars_selected))) {
  columns <- c(colNames[seq(j)])
  tr_ctr <- trainControl(method = "none")
  dataForTrt <- data.frame(cbind(subset(trainingDataset, select = columns),
                                labelsTrain))
  colnames(dataForTrt)[seq(j)] <- make.names(columns)

  nn_model <- train(
    labelsTrain ~ .,
    data = dataForTrt,
    method = "nnet",
    tuneGrid = data.frame(size = bestParameters$size, decay =
                          bestParameters$decay),
    maxit = maxit,
    trControl = tr_ctr,
    preprocess = c("center", "scale"),
    trace = FALSE
  )

  testX <- subset(testDataset, select = columns)
  predicts <- predict(nn_model, newdata = testX)

  cfMatList[[i]] <- confusionMatrix(predicts, labelsTest)
  acc_cv[i, j] <- cfMatList[[i]]$overall[[1]]

  if (length(levels(labelsTrain)) == 2) {
    sens <- cfMatList[[i]]$byClass[[1]]
    spec <- cfMatList[[i]]$byClass[[2]]
  }
}

```

```

    f1 <- cfMatList[[i]]$byClass[[7]]
  } else {
    sens <- mean(cfMatList[[i]]$byClass[, 1])
    spec <- mean(cfMatList[[i]]$byClass[, 2])
    f1 <- mean(cfMatList[[i]]$byClass[, 7])
  }

  sens_cv[i, j] <- sens
  spec_cv[i, j] <- spec
  f1_cv[i, j] <- f1

  if (is.na(sens_cv[i, j])) sens_cv[i, j] <- 0
  if (is.na(spec_cv[i, j])) spec_cv[i, j] <- 0
  if (is.na(f1_cv[i, j])) f1_cv[i, j] <- 0
}
}

meanAcc <- colMeans(acc_cv)
names(meanAcc) <- colnames(acc_cv)
sdAcc <- apply(acc_cv, 2, sd)
accuracyInfo <- list(meanAcc, sdAcc)
names(accuracyInfo) <- c("meanAccuracy", "standardDeviation")

meanSens <- colMeans(sens_cv)
names(meanSens) <- colnames(sens_cv)
sdSens <- apply(sens_cv, 2, sd)
sensitivityInfo <- list(meanSens, sdSens)
names(sensitivityInfo) <- c("meanSensitivity", "standardDeviation")

meanSpec <- colMeans(spec_cv)
names(meanSpec) <- colnames(spec_cv)
sdSpec <- apply(spec_cv, 2, sd)
specificityInfo <- list(meanSpec, sdSpec)
names(specificityInfo) <- c("meanSpecificity", "standardDeviation")

meanF1 <- colMeans(f1_cv)
names(meanF1) <- colnames(f1_cv)
sdF1 <- apply(f1_cv, 2, sd)
F1Info <- list(meanF1, sdF1)

```

```

names(F1Info) <- c("meanF1", "standardDeviation")

cat("Neural network classification done successfully!\n")
results_cv <- list(cfMatList, accuracyInfo, sensitivityInfo, specificityInfo,
                  F1Info, bestParameters)
names(results_cv) <- c("cfMats", "accuracyInfo", "sensitivityInfo",
                      "specificityInfo", "F1Info", "bestParameters")
invisible(results_cv)
}

```

```

nn_trn_mrmr <- nn_trn(MLMatrix, MLLabels, vars_selected = FSRankingmRMR[1:10],
                     numFold = 5)
save(nn_trn_mrmr, file = "nn_trn_mrmr")

```

```
load(file = "nn_trn_mrmr")
```

Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

```

nn_results_mrmr_trn <- rbind(nn_trn_mrmr$F1Info$meanF1[1:10],
                             nn_trn_mrmr$accuracyInfo$meanAccuracy, nn_trn_mrmr$sensitivityInfo$meanSensitivity,
                             nn_trn_mrmr$specificityInfo$meanSpecificity)
save(nn_results_mrmr_trn, file = "nn_results_mrmr_trn")

```

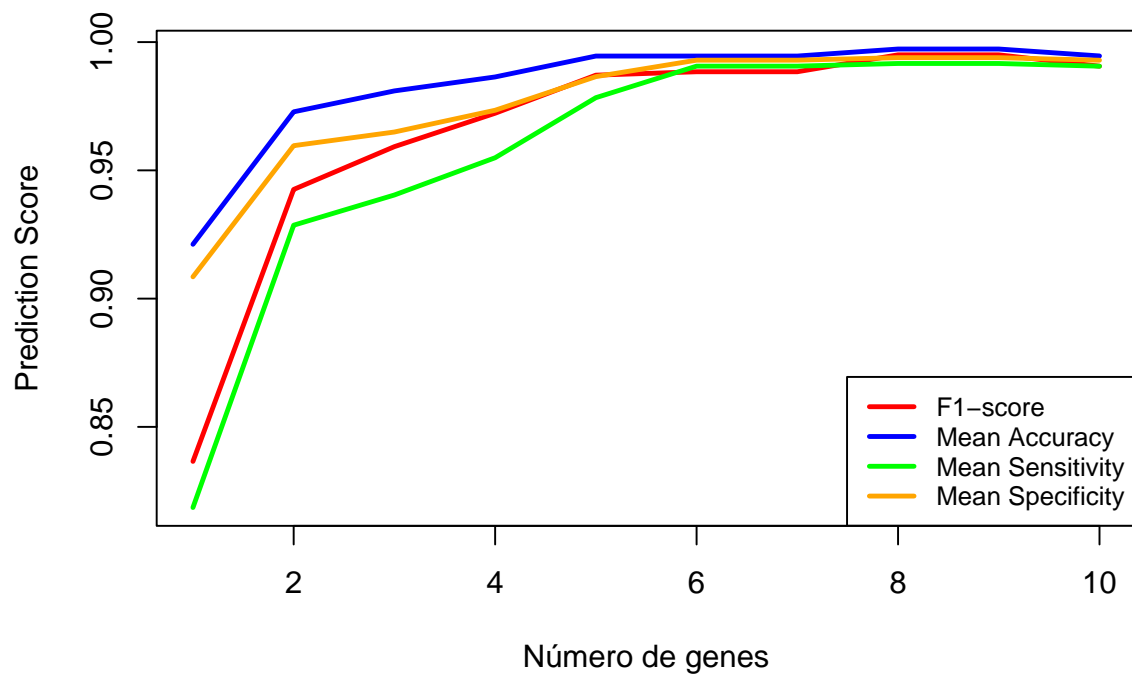
```
load(file = "nn_results_mrmr_trn")
```

Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```

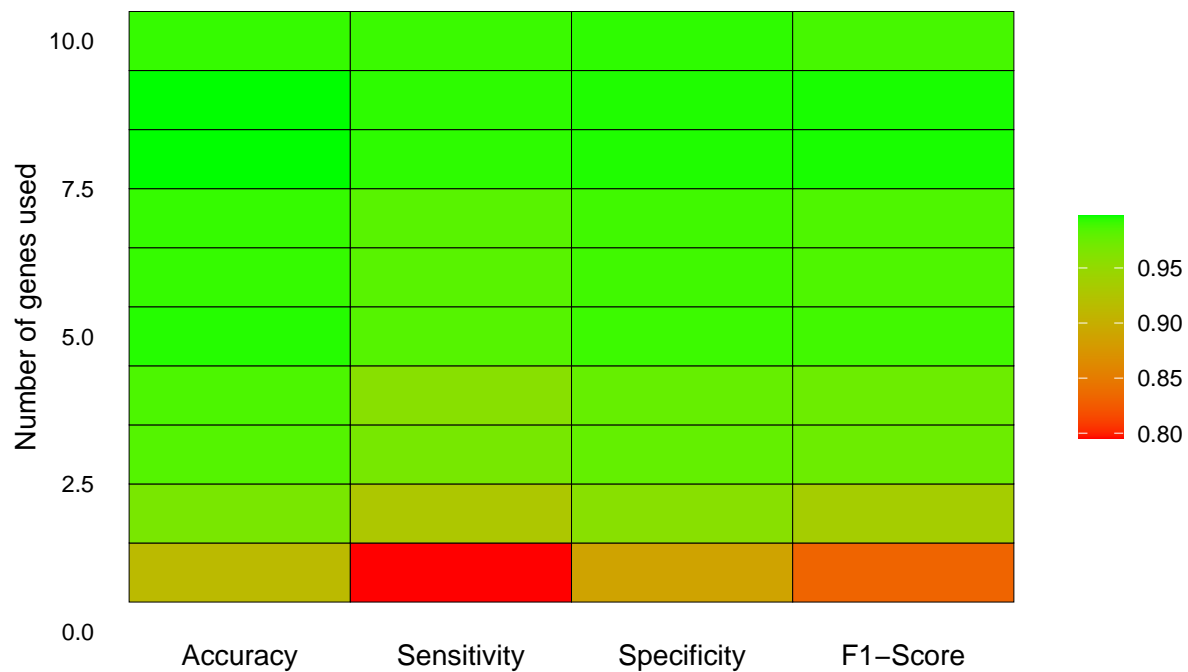
dataPlot(nn_results_mrmr_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
           "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
         xlab="Número de genes", ylab="Prediction Score")

```



Mostramos un mapa de calor de los resultados:

```
dataPlot(nn_trn_mrmr, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por mRMR y los `best_params` obtenidos durante el entrenamiento:

```
mn_test <- function(train, labelsTrain, test, labelsTest, vars_selected, bestParameters){

  if (!is.data.frame(train) && !is.matrix(train)) {
    stop("The train argument must be a dataframe or a matrix.")
  }

  if (dim(train)[1] != length(labelsTrain)) {
    stop("The length of the rows of the argument train must be the same as the
          length of the labelsTrain. Please, ensure that the rows are the samples
          and the columns are the variables.")
  }

  if (!is.character(labelsTrain) && !is.factor(labelsTrain)) {
    stop("The class of the labelsTrain parameter must be character vector or
          factor.")
  }
  if (is.character(labelsTrain)) { labelsTrain <- as.factor(labelsTrain) }

  if (!is.character(labelsTest) && !is.factor(labelsTest)) {
    stop("The class of the labelsTest parameter must be character vector or
          factor.")
  }
  if (is.character(labelsTest)) { labelsTest <- as.factor(labelsTest) }

  if (!is.data.frame(test) && !is.matrix(test)) {
    stop("The test argument must be a dataframe or a matrix.")
  }

  if (dim(test)[1] != length(labelsTest)) {
    stop("The length of the rows of the argument test must be the same as the
          length of the labelsTest. Please, ensure that the rows are the samples
          and the columns are the variables.")
  }

  train <- as.data.frame(apply(train, 2, as.double))
  train <- train[, vars_selected]
```

```

test <- as.data.frame(apply(test, 2, as.double))
test <- test[, vars_selected]

train <- vapply(train, function(x) {
  max <- max(x)
  min <- min(x)
  if (max > min) {
    x <- ((x - min) / (max - min)) * 2 - 1
  }
  else {
    x
  }}, double(nrow(train)))

train <- as.data.frame(train)

test <- vapply(test, function(x) {
  max <- max(x)
  min <- min(x)
  if (max > min) {
    x <- ((x - min) / (max - min)) * 2 - 1
  }
  else {
    x
  }}, double(nrow(test)))

test <- as.data.frame(test)

colNames <- colnames(train)

accVector <- double()
sensVector <- double()
specVector <- double()
f1Vector <- double()
cfMatList <- list()

# Firstly with 1 variable
cat(paste("Testing with ", 1, " variables...\n", sep = ""))

columns <- c(colNames[1])

```



```

tr_ctr <- trainControl(method = "none")
dataForTrt <- data.frame(cbind(subset(train, select = columns), labelsTrain))
colnames(dataForTrt)[seq(1)] <- make.names(columns)

nn_mod <- train(labelsTrain ~ .,
               data = dataForTrt,
               method = 'nnet',
               metric = 'Accuracy',
               preProc = c("center", "scale"),
               tuneGrid = data.frame(.size = bestParameters[1], .decay =
                                     bestParameters[2]),
               trace = FALSE)

testX <- subset(test, select = columns)
unkX <- testX
colnames(unkX) <- make.names(colnames(testX))
colnames(testX) <- make.names(colnames(testX))
predicts <- predict(nn_mod, newdata = testX)

cfMat <- confusionMatrix(predicts, labelsTest)
if (length(levels(labelsTrain)) == 2) {
  sens <- cfMat$byClass[[1]]
  spec <- cfMat$byClass[[2]]
  f1 <- cfMat$byClass[[7]]
} else {
  sens <- mean(cfMat$byClass[, 1])
  spec <- mean(cfMat$byClass[, 2])
  f1 <- mean(cfMat$byClass[, 7])
}

cfMatList[[1]] <- cfMat
accVector[1] <- cfMat$overall[[1]]
sensVector[1] <- sens
specVector[1] <- spec
f1Vector[1] <- f1

for (i in c(2:dim(train)[2])) {

  cat(paste("Testing with ", i, " variables...\n", sep = ""))

```

```

columns <- c(colNames[seq(i)])
tr_ctr <- trainControl(method = "none")
dataForTrt <- data.frame(cbind(subset(train, select = columns), labelsTrain))
colnames(dataForTrt)[seq(i)] <- make.names(columns)

nn_mod <- train(labelsTrain ~ .,
               data = dataForTrt,
               method = 'nnet',
               metric = 'Accuracy',
               preProc = c("center", "scale"),
               tuneGrid = data.frame(.size = bestParameters[1], .decay =
                                     bestParameters[2]),
               trace = FALSE)

testX <- subset(test, select = columns)
unkX <- testX
colnames(unkX) <- make.names(colnames(testX))
colnames(testX) <- make.names(colnames(testX))
predicts <- predict(nn_mod, newdata = testX)

cfMat <- confusionMatrix(predicts, labelsTest)
if (length(levels(labelsTrain)) == 2) {
  sens <- cfMat$byClass[[1]]
  spec <- cfMat$byClass[[2]]
  f1 <- cfMat$byClass[[7]]
} else {
  sens <- mean(cfMat$byClass[, 1])
  spec <- mean(cfMat$byClass[, 2])
  f1 <- mean(cfMat$byClass[, 7])
}

cfMatList[[i]] <- cfMat
accVector[i] <- cfMat$overall[[1]]
sensVector[i] <- sens
specVector[i] <- spec
f1Vector[i] <- f1
}

cat("Classification done successfully!\n")

```

```

names(accVector) <- vars_selected
names(sensVector) <- vars_selected
names(specVector) <- vars_selected
names(f1Vector) <- vars_selected

results <- list(cfMatList, accVector, sensVector, specVector, f1Vector)
names(results) <- c("cfMats","accVector","sensVector","specVector","f1Vector")
invisible(results)

}

```

Realizamos un pequeño ajuste a la variable XTest que necesitaremos para la parte del test.

```

nn_trn_mrmr$bestParameters
nn_test_mrmr <- nn_test(MLMatrix, MLLabels, t(XTest), YTest, vars_selected =
                      vars_selected[1:10], bestParameters = nn_trn_mrmr$bestParameters)

save(nn_test_mrmr, file = "nn_test_mrmr")

```

```
load(file = "nn_test_mrmr" )
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```

nn_results_mrmr_test <- rbind(nn_trn_mrmr$F1Info$meanF1[1:10],nn_test_mrmr$f1Vector,
                             nn_trn_mrmr$accuracyInfo$meanAccuracy, nn_test_mrmr$accVector)
save(nn_results_mrmr_test,file = "nn_results_mrmr_test")

```

```
load(file = "nn_results_mrmr_test")
```

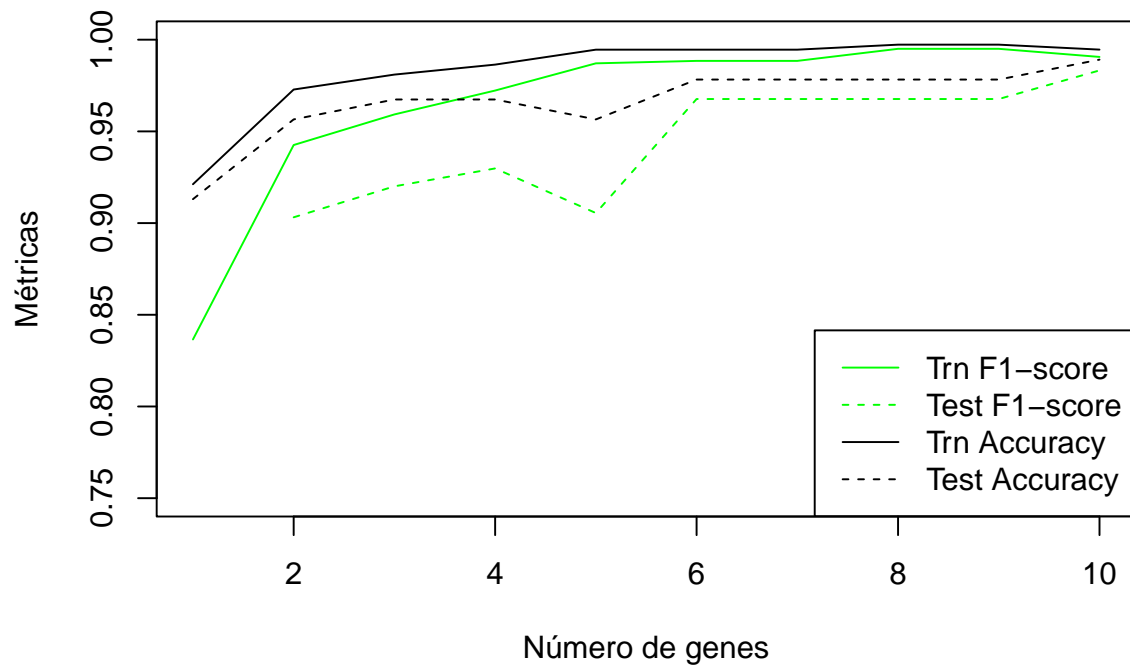
Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- nn_results_mrmr_test[1,1:10]
Test_F1 <- nn_results_mrmr_test[2,1:10]
Trn_Acc <- nn_results_mrmr_test[3,1:10]
Test_Acc <- nn_results_mrmr_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")

```

```
lines(num_genes, Test_Acc,"1", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
      col=c("green","green","black","black"),lty=c(1,2,1,2) )
```



Mostramos la matriz de confusión en el conjunto de test:

```
dataPlot(nn_test_mrmr$cfMats[[2]]$table, MLLabels, mode = "confusionMatrix")
```

Reference	CHOL	7	0	0
	Healthy	1	10	0
	LIHC	3	0	71
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
95.652	90.316	95.618	98.431

Este algoritmo de clasificación Redes Neuronales presenta picos en la primera gráfica como podemos observar por lo que el mejor número de genes a elegir sería **5**, ya que aproximadamente dan una explicabilidad del 95% aproximadamente. Otra opción sería utilizar 8 genes ya tendría mayor explicabilidad pero con **5 genes** parece explicar bastante bien.

3.4.2 Algoritmo de selección de características: RF

En esta sección se utilizará el método de selección de características Random Forest (RF). Se seguirán los mismos pasos aplicados anteriormente con mRMR para analizar los resultados obtenidos tanto en el entrenamiento como en el test, y así comparar su rendimiento.

En primer lugar, realizamos el entrenamiento del modelo Redes neuronales utilizando las 10 características seleccionadas por el método RF:

```
nn_trn_rf <- nn_trn(MLMatrix, MLLabels, vars_selected = FSRankingRF[1:10],
                    numFold = 5)
save(nn_trn_rf, file = "nn_trn_rf")
```

```
load(file = "nn_trn_rf")
```

Se combinan las métricas principales (F1-score, Accuracy, Sensitivity y Specificity) para los 10 primeros genes seleccionados por RF en una sola matriz, lo que facilita su visualización.

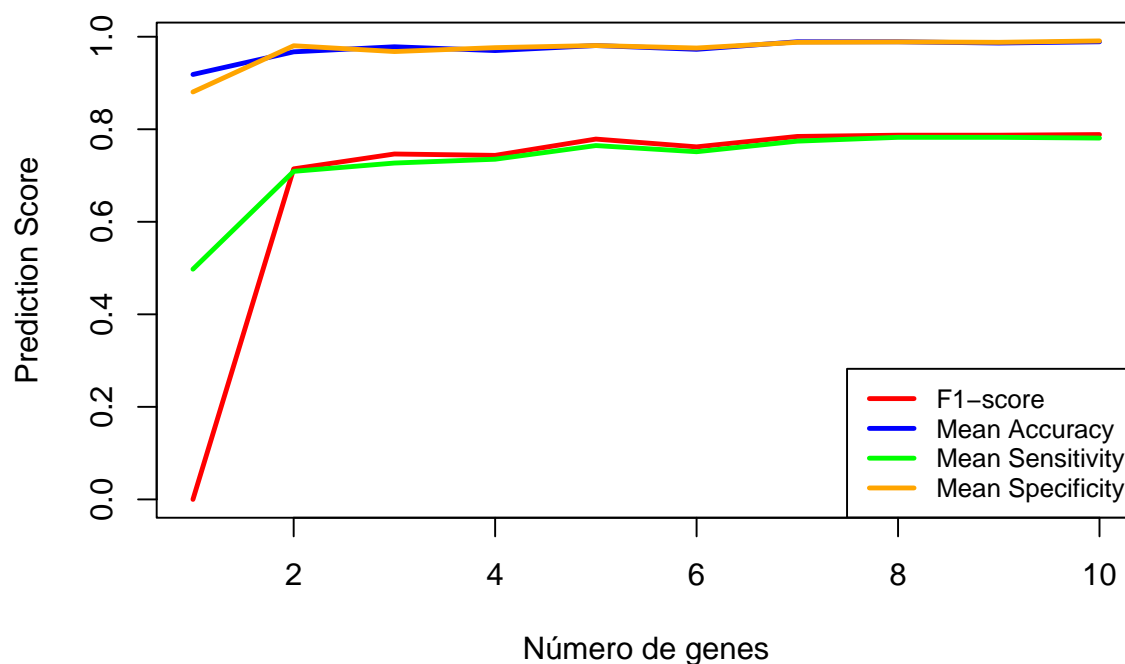
```
nn_results_rf_trn <- rbind(nn_trn_rf$F1Info$meanF1[1:10],
  nn_trn_rf$accuracyInfo$meanAccuracy, nn_trn_rf$sensitivityInfo$meanSensitivity,
  nn_trn_rf$specificityInfo$meanSpecificity)
```

```
save(nn_results_rf_trn,file = "nn_results_rf_trn")
```

```
load(file = "nn_results_rf_trn")
```

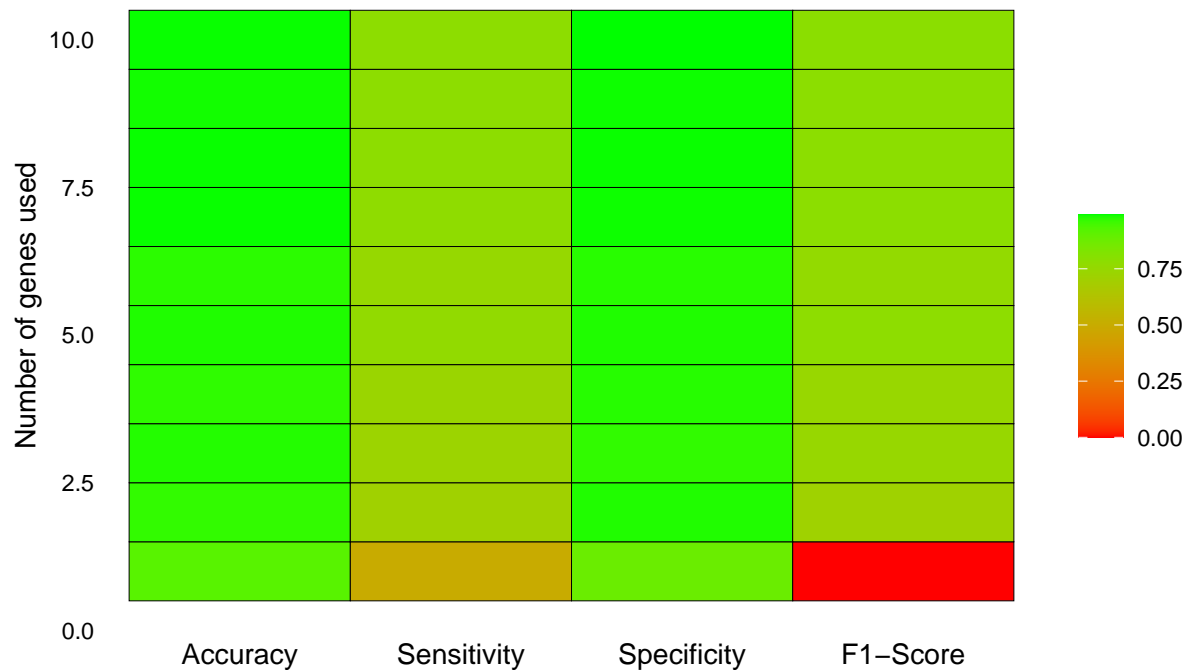
A continuación, se crea una gráfica que muestre el rendimiento de cada métrica en función del número de genes.

```
dataPlot(nn_results_rf_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Se mostrará un mapa de calor con los resultados globales del entrenamiento:

```
dataPlot(nn_trn_rf, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por RF y los best params obtenidos durante el entrenamiento:

```
nn_test_rf <- nn_test(MLMatrix, MLLabels, t(XTest), YTest, vars_selected=
                      FSRankingRF[1:10],bestParameters=nn_trn_rf$bestParameters)

save(nn_test_rf,file = "nn_test_rf")
```

```
load(file = "nn_test_rf")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
nn_results_rf_test <- rbind(nn_trn_rf$F1Info$meanF1[1:10], nn_test_rf$f1Vector,
                           nn_trn_rf$accuracyInfo$meanAccuracy, nn_test_rf$accVector)

save(nn_results_rf_test,file = "nn_results_rf_test")
```

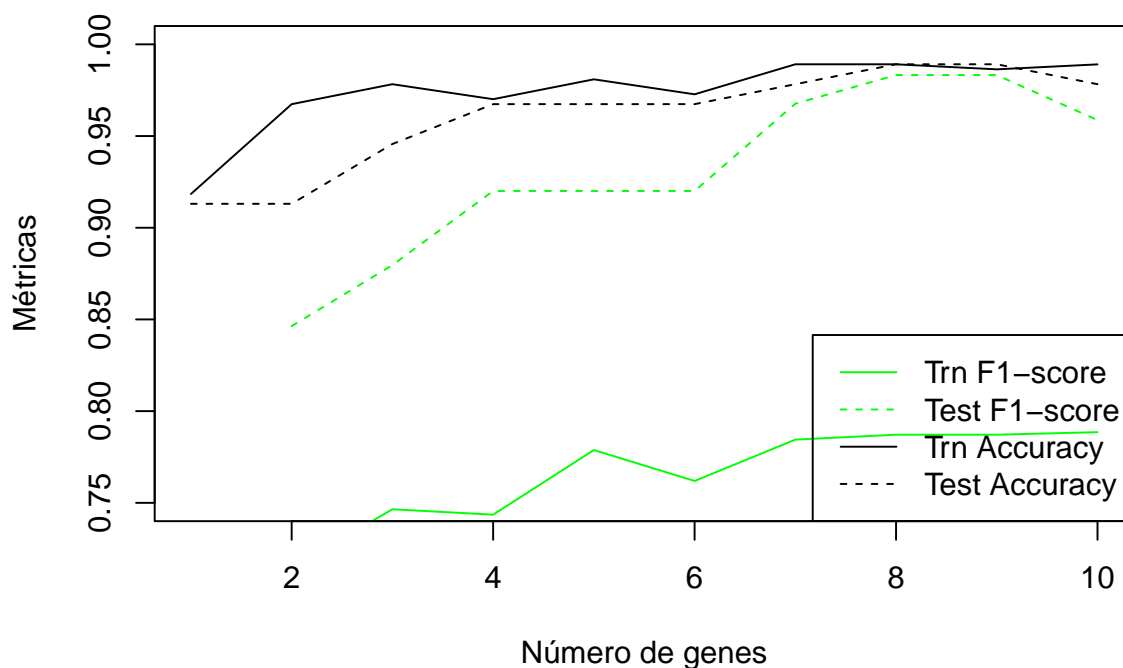
```
load(file = "nn_results_rf_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- nn_results_rf_test[1,1:10]
Test_F1 <- nn_results_rf_test[2,1:10]
Trn_Acc <- nn_results_rf_test[3,1:10]
Test_Acc <- nn_results_rf_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.75,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Por último, se mostrará la matriz de confusión:

```

dataPlot(nn_test_rf$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```


Reference	CHOL	6	1	0
	Healthy	0	11	0
	LIHC	3	1	70
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy	F1	Sensitivity	Specificity
94.565	87.963	93.436	98

En el algoritmo de selección de características Random Forest, se aprecia un pico en el número de genes igual a 5, lo que indica una alta capacidad explicativa con cuatro genes, pero si queremos más precisión lo suyo sería elegir el número de genes igual a 7 ya que estos superan el 95% de precisión. Por tanto, el número de genes elegidos será **5**.

3.4.3 Algoritmo de selección de características: DA

Esta será una de las últimas pruebas que se hará para el algoritmo de Redes neuronales, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo Red neuronal utilizando las 10 características seleccionadas por el método DA:

```
nn_trn_da <- nn_trn(MLMatrix, MLLabels, vars_selected = names(FSRankingDA[1:10]),
                    numFold = 5)
```

```
save(nn_trn_da,file = "nn_trn_da")
```

```
load(file = "nn_trn_da")
```

Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

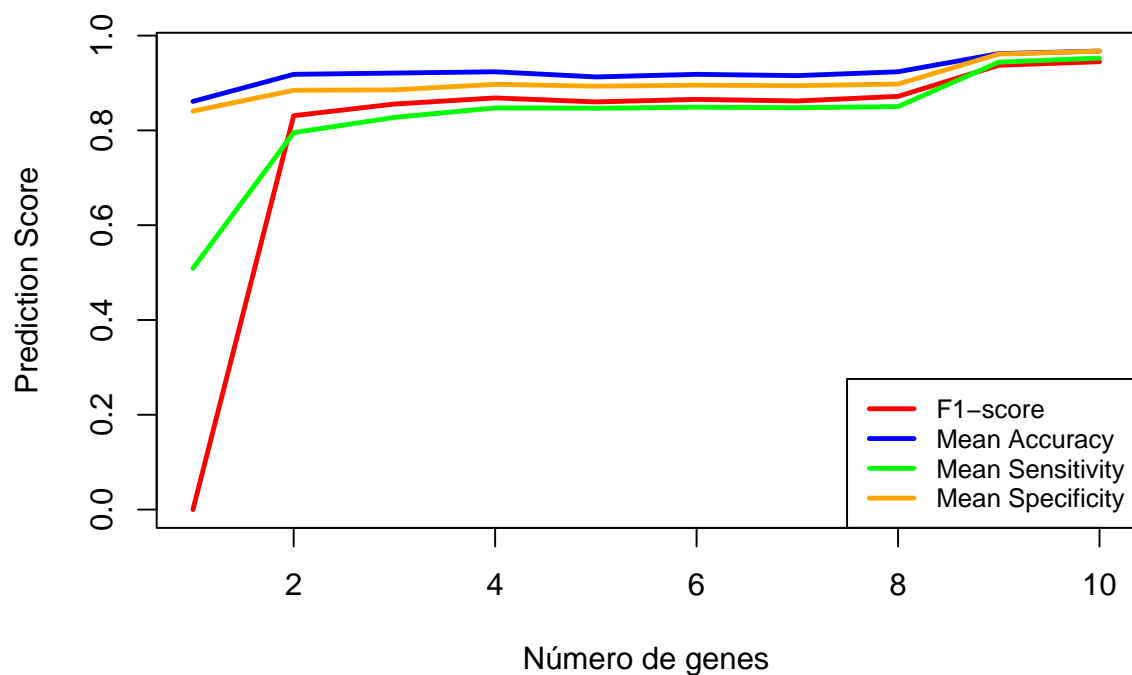
```
nn_results_da_trn <- rbind(nn_trn_da$F1Info$meanF1[1:10],
  nn_trn_da$accuracyInfo$meanAccuracy, nn_trn_da$sensitivityInfo$meanSensitivity,
  nn_trn_da$specificityInfo$meanSpecificity)
```

```
save(nn_results_da_trn,file = "nn_results_da_trn")
```

```
load(file = "nn_results_da_trn")
```

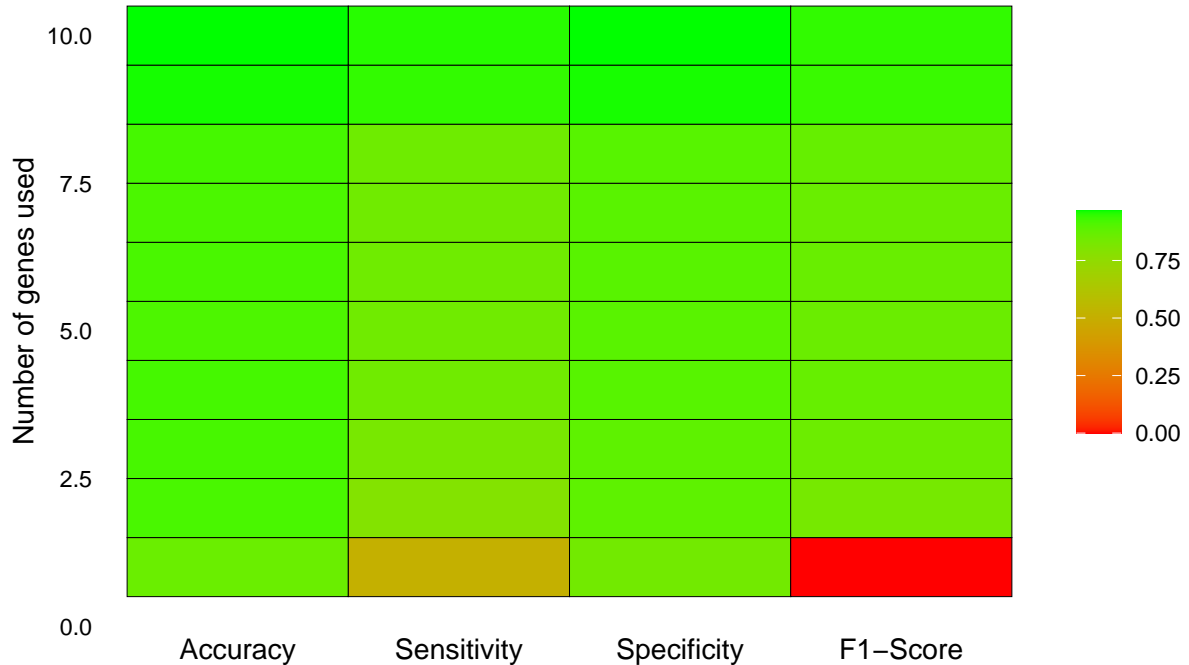
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(nn_results_da_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados:

```
dataPlot(nn_trn_da, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por DA y los best params obtenidos durante el entrenamiento:

```
nn_test_da <- nn_test(MLMatrix, MLLabels, t(XTest), YTest, vars_selected=
  names(FSRankingDA[1:10]), bestParameters=nn_trn_da$bestParameters)
save(nn_test_da, file = "nn_test_da")
```

```
load(file = "nn_test_da")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
nn_results_da_test <- rbind(nn_trn_da$F1Info$meanF1[1:10], nn_test_da$f1Vector,
  nn_trn_da$accuracyInfo$meanAccuracy, nn_test_da$accVector)
save(nn_results_da_test, file = "nn_results_da_test")
```

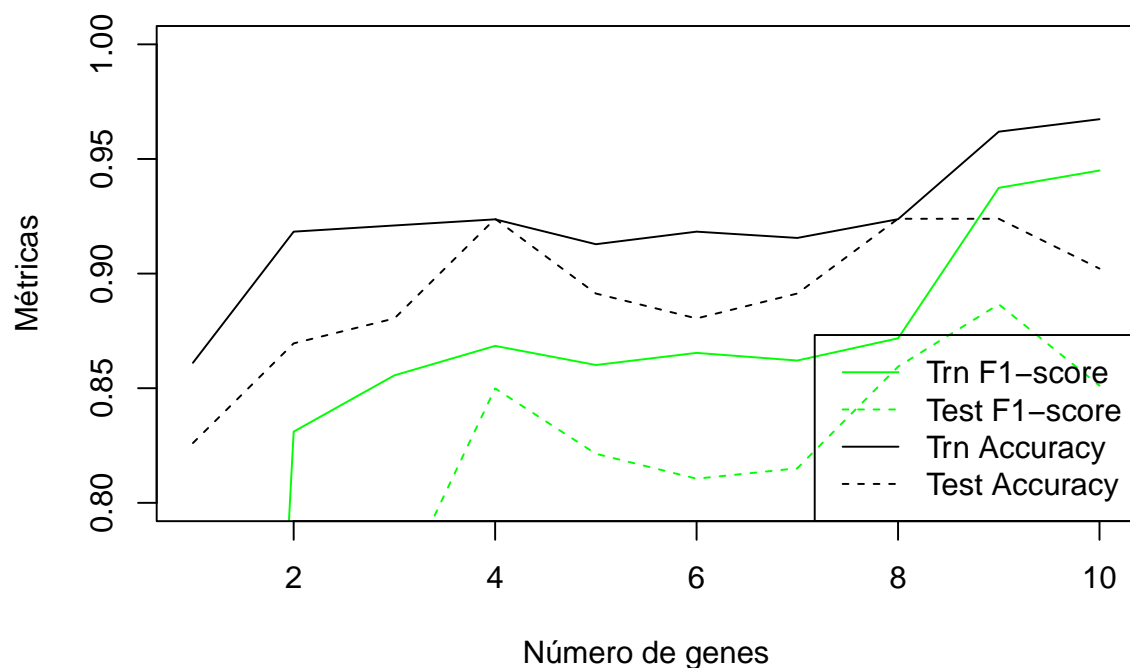
```
load(file = "nn_results_da_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- nn_results_da_test[1,1:10]
Test_F1 <- nn_results_da_test[2,1:10]
Trn_Acc <- nn_results_da_test[3,1:10]
Test_Acc <- nn_results_da_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.80,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(nn_test_da$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```

Reference	CHOL	6	1	0
	Healthy	1	5	5
	LIHC	0	4	70
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 88.043	F1 75.764	Sensitivity 75.254	Specificity 88.291
---------------------------	---------------------	------------------------------	------------------------------

En este algoritmo de selección de características, se puede observar que el número de genes con mayor explicabilidad es 9, ya que alcanza casi un 90%. Sin embargo, son bastantes genes y con un número de genes igual a 4 se puede observar que hay una explicabilidad alrededor del 85% por lo que será el elegido.

3.4.4 Algoritmo de selección de características: Lasso

Esta será la última prueba que se hará para el algoritmo de Red neuronal, utilizando un método de selección de características diferente. Se seguirán los mismos pasos que anteriormente.

En primer lugar, realizamos el entrenamiento del modelo de Red neuronal utilizando las 10 características seleccionadas por el método Lasso:

```
nn_trn_lasso <- nn_trn(MLMatrix, MLLabels, vars_selected = FSRankingLasso[1:10],
                      numFold = 5)
```

```
save(nn_trn_lasso,file = "nn_trn_lasso")
```

```
load(file = "nn_trn_lasso")
```

Calculamos las métricas de evaluación F1-score, Accuracy, Sensitivity y Specificity y almacenamos los resultados:

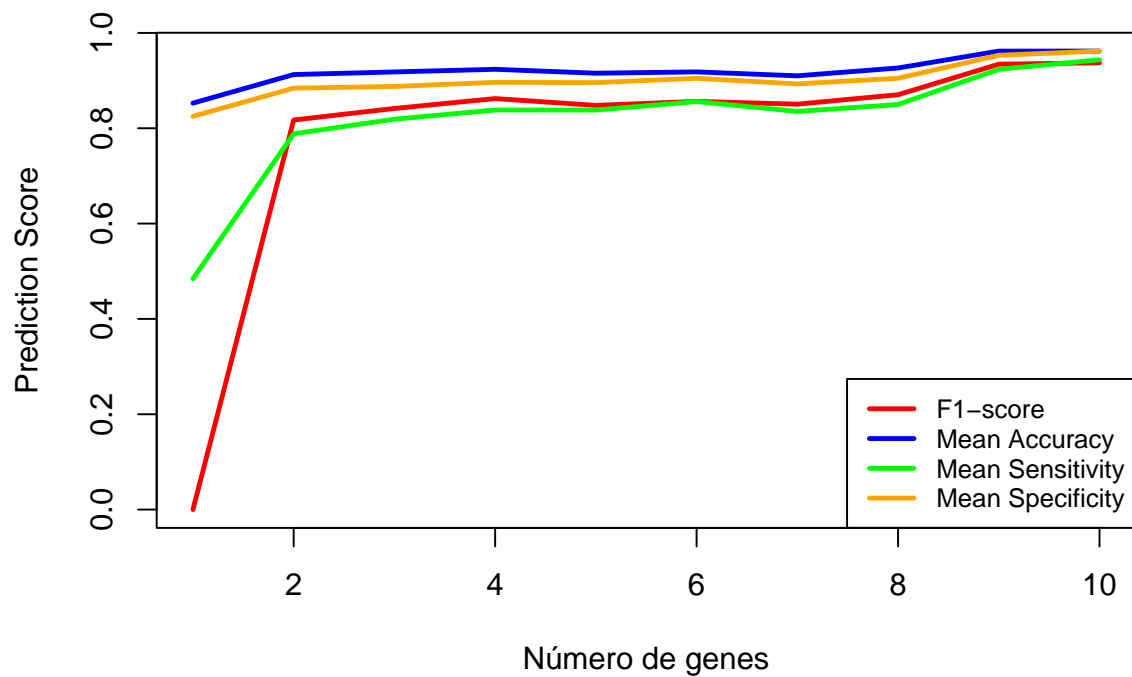
```
nn_results_lasso_trn <- rbind(nn_trn_lasso$F1Info$meanF1[1:10],
  nn_trn_lasso$accuracyInfo$meanAccuracy, nn_trn_lasso$sensitivityInfo$meanSensitivity,
  nn_trn_lasso$specificityInfo$meanSpecificity)
```

```
save(nn_results_lasso_trn,file = "nn_results_lasso_trn")
```

```
load(file = "nn_results_lasso_trn")
```

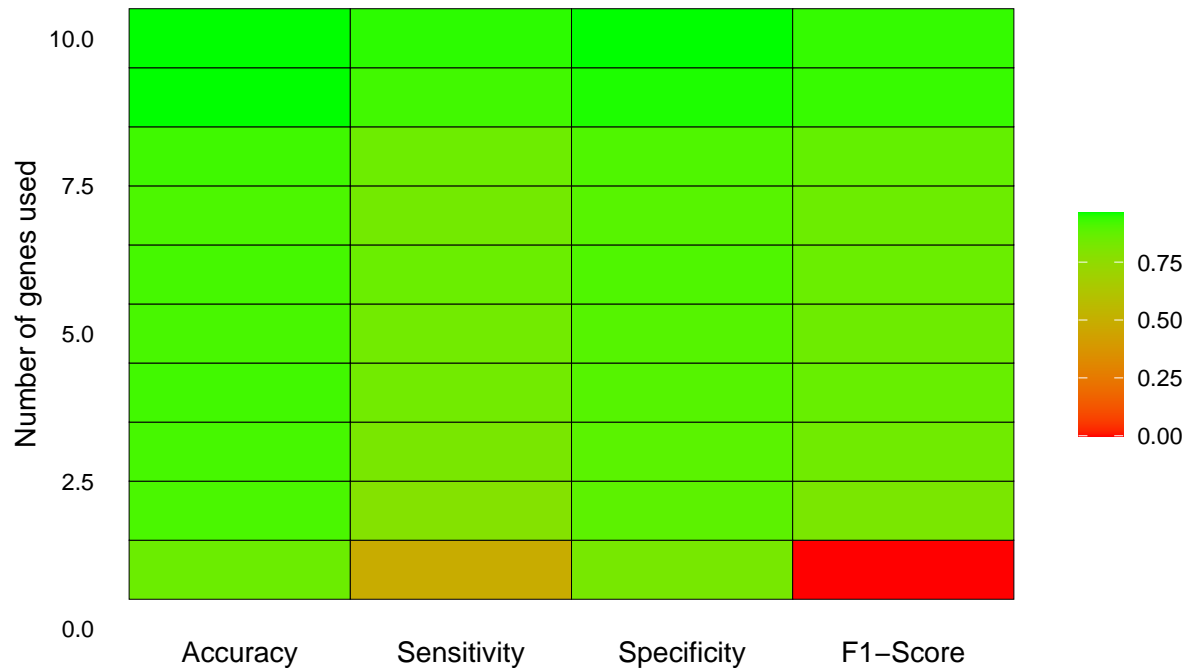
Visualizamos las métricas obtenidas mediante gráficos para interpretar los resultados:

```
dataPlot(nn_results_lasso_trn, MLLabels, legend = c("F1-score", "Mean Accuracy",
  "Mean Sensitivity", "Mean Specificity"), mode = "classResults",
  xlab="Número de genes", ylab="Prediction Score")
```



Mostramos un mapa de calor de los resultados:

```
dataPlot(nn_trn_lasso, MLLabels, mode = "heatmapResults")
```



A continuación, evaluamos el modelo utilizando el conjunto de test con las características seleccionadas por Lasso y los best params obtenidos durante el entrenamiento:

```
nn_test_lasso <- nn_test(MLMatrix, MLLabels, t(XTest), YTest, vars_selected=
  FSRankingLasso[1:10], bestParameters=nn_trn_lasso$bestParameters)
save(nn_test_lasso, file = "nn_test_lasso")
```

```
load(file = "nn_test_lasso")
```

Se recopilan las métricas obtenidas en el entrenamiento y en el test (Accuracy y F1-score):

```
nn_results_lasso_test <- rbind(nn_trn_lasso$F1Info$meanF1[1:10],
  nn_test_lasso$f1Vector, nn_trn_lasso$accuracyInfo$meanAccuracy,
  nn_test_lasso$accVector)
save(nn_results_lasso_test, file = "nn_results_lasso_test")
```

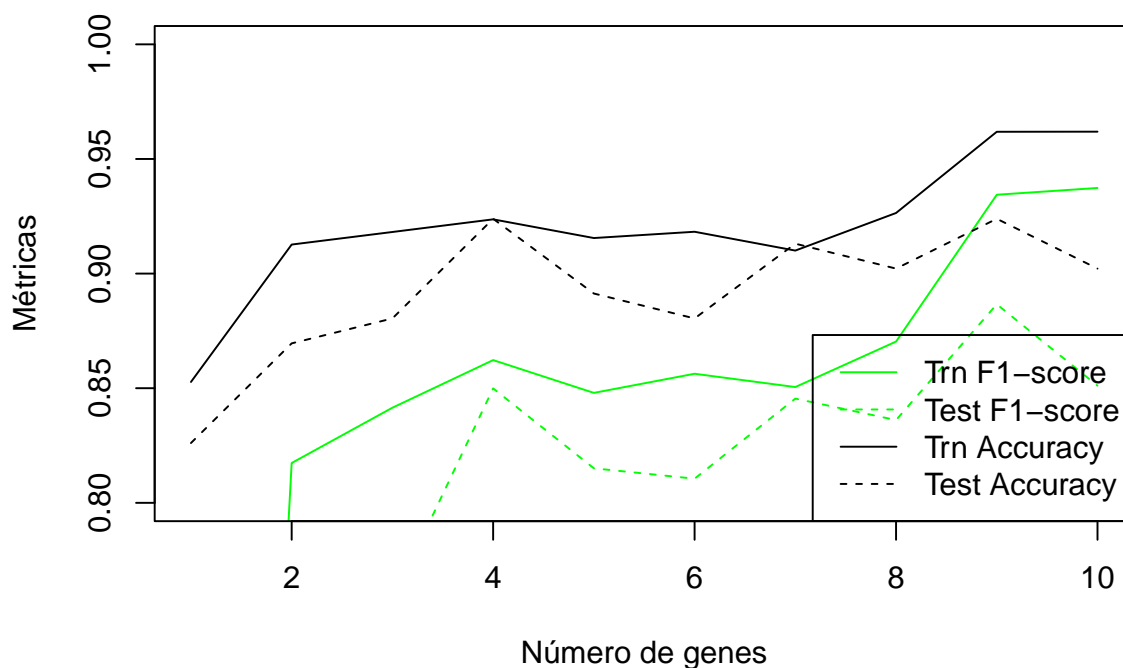
```
load(file = "nn_results_lasso_test")
```

Visualizamos los resultados comparando train y test:

```

num_genes <- 1:10
Trn_F1 <- nn_results_lasso_test[1,1:10]
Test_F1 <- nn_results_lasso_test[2,1:10]
Trn_Acc <- nn_results_lasso_test[3,1:10]
Test_Acc <- nn_results_lasso_test[4,1:10]
plot(num_genes, Trn_F1,"l", col = "green",ylim=c(0.80,1.0),ylab = "Métricas",
     xlab = "Número de genes")
lines(num_genes, Test_F1,"l",col = "green" ,lty = 2)
lines(num_genes, Trn_Acc,"l")
lines(num_genes, Test_Acc,"l", lty = 2)
legend("bottomright",c("Trn F1-score","Test F1-score","Trn Accuracy","Test Accuracy"),
     col=c("green","green","black","black"),lty=c(1,2,1,2) )

```



Mostramos la matriz de confusión en el conjunto de test:

```

dataPlot(nn_test_lasso$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

```


Reference	CHOL	6	1	0
	Healthy	1	5	5
	LIHC	0	4	70
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 88.043	F1 75.764	Sensitivity 75.254	Specificity 88.291
---------------------------	---------------------	------------------------------	------------------------------

En este algoritmo de selección de características, se puede observar que el número de genes con mayor explicabilidad es 9, ya que alcanza casi un 95%. Sin embargo, son bastantes genes y con un número de genes igual a 4 se puede observar que hay picos en las métricas de TRN y presenta una explicabilidad alrededor del 85% por lo que será el elegido.

3.4.5 Conclusiones

Tras analizar los resultados obtenidos con diferentes métodos de selección de características, considero que la mejor opción es utilizar **mRMR** con un número de **5 genes** seleccionados. Con esta configuración, se encuentra entre alrededor del 95% de predicción, alcanzando además los valores más altos de **Accuracy** y **F1-score** en comparación con los demás algoritmos de selección de características.

3.5 Elecciones finales

Tras realizar estas pruebas para cada uno de los algoritmos de clasificación, finalmente los resultados han sido los siguientes:

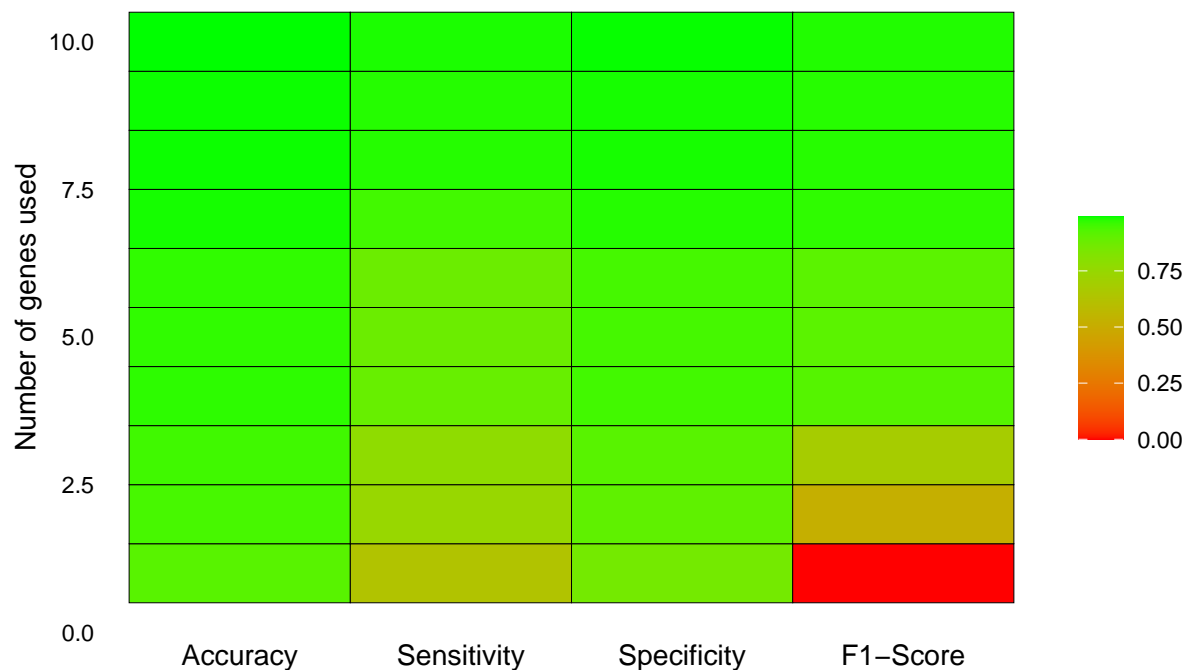
- El algoritmo **k-NN** ha obtenido mejores métricas con el algoritmo de selección de características **Random Forest**.
- El algoritmo **Random Forest** ha obtenido mejores métricas con el algoritmo de selección de características **mRMR**.

- El algoritmo **SVM** ha obtenido mejores métricas con el algoritmo de selección de características **Random Forest**.
- El algoritmo **Red neuronal** ha obtenido mejores métricas con al algoritmo de selección de características **mRMR**

En este caso, se han obtenido mejores métricas de **Accuracy** y **F1-score** para cada uno de los tres algoritmos de clasificación utilizando el algoritmo de selección de características **Random Forest** por lo que ha sido seleccionado. Además, entre los algoritmos de clasificación, el que ha presentado mejores métricas ha sido **SVM**, por lo que será el utilizado.

Mostraremos de nuevo los resultados de este modelo, en concreto del **heatmap** y de la **matriz de confusión del conjunto de test**:

```
dataPlot(svm_trn_rf, MLLabels, mode = "heatmapResults")
```



```
dataPlot(svm_test_rf$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")
```

Reference	CHOL	6	1	0
	Healthy	0	11	0
	LIHC	0	1	73
		CHOL	Healthy	LIHC
		Prediction		

DETAILS

Accuracy 97.826	F1 94.431	Sensitivity 94.788	Specificity 99.177
---------------------------	---------------------	------------------------------	------------------------------

En el siguiente apartado se realizará una validación cruzada de 5 particiones (5CV) en el dataset completo, utilizando el algoritmo de selección de características y el clasificador escogido. Se evalúa el rendimiento medio del modelo (en entrenamiento y test) a medida que aumenta el número de genes seleccionados, mostrando los resultados en una gráfica que refleja esta evolución. Además, se presentan los 5 rankings de genes obtenidos en cada fold de la validación cruzada y, finalmente, se selecciona la huella final de genes, que corresponde al conjunto óptimo basado en el consenso entre rankings y el rendimiento observado en la gráfica.

4 Evolución del rendimiento según el número de genes utilizando 5-CV con el clasificador seleccionado

En este apartado se seleccionará un algoritmo de clasificación junto con un algoritmo de selección de características óptimo. Como se detalla en la sección **3.4 Elecciones finales**, se identificaron los mejores algoritmos de selección de características para cada clasificador, basándose en las métricas de **Accuracy** y **F1-score**.

De entre las opciones evaluadas, se ha elegido el clasificador **SVM** junto con el algoritmo de selección de características **Random Forest**, ya que obtuvo las mejores métricas: un **Accuracy** de **97.826** y un **F1-score** de **94.431**, superando a los demás métodos considerados.

```
# APARTADO VALIDACIÓN CRUZADA
nVarsMAX <- 20 # Número máximo de genes
nfolds <- 5    # Número de particiones para 5CV
set.seed(7) # Semilla propia

# Inicialización de matrices
ACCTrnRF <- matrix(0, nfolds, nVarsMAX)
ACCTestRF <- matrix(0, nfolds, nVarsMAX)
rankingSRF <- matrix(0, nfolds, nVarsMAX)

# Generar particiones estratificadas
foldIdx <- cvGenStratified(LABELS, nfolds)

for (particion in seq(1, nfolds)) {
  cat("Partición", particion, "\n")

  indexTest <- which(foldIdx == particion)
  indexTrn <- setdiff(seq(1, length(LABELS)), indexTest)

  XTrn <- t(MATRIZ)[indexTrn, ]
  YTrn <- LABELS[indexTrn]
  XTest <- t(MATRIZ)[indexTest, ]
  YTest <- LABELS[indexTest]

  rankingRF <- featureSelection(XTrn, YTrn, mode = "rf",
                               vars_selected = rownames(DEGsMatrix))
  rankingSRF[particion, ] <- rankingRF[1:nVarsMAX]
```

```

for (i in 1:nVarsMAX) {
  vars_selected <- rankingRF[1:i]
  vars_selected <- vars_selected[vars_selected %in% colnames(XTrn)]

  if (length(vars_selected) == 0) {
    stop("No hay variables seleccionadas válidas en esta iteración")
  }

  svm_mod <- svm(XTrn[, vars_selected, drop = FALSE], as.factor(YTrn),
    kernel= "radial")
  pred_train <- predict(svm_mod, XTrn[, vars_selected, drop = FALSE])
  confMatrixTrn <- confusionMatrix(as.factor(pred_train), as.factor(YTrn))
  ACCTrnRF[particion, i] <- confMatrixTrn$overall["Accuracy"]

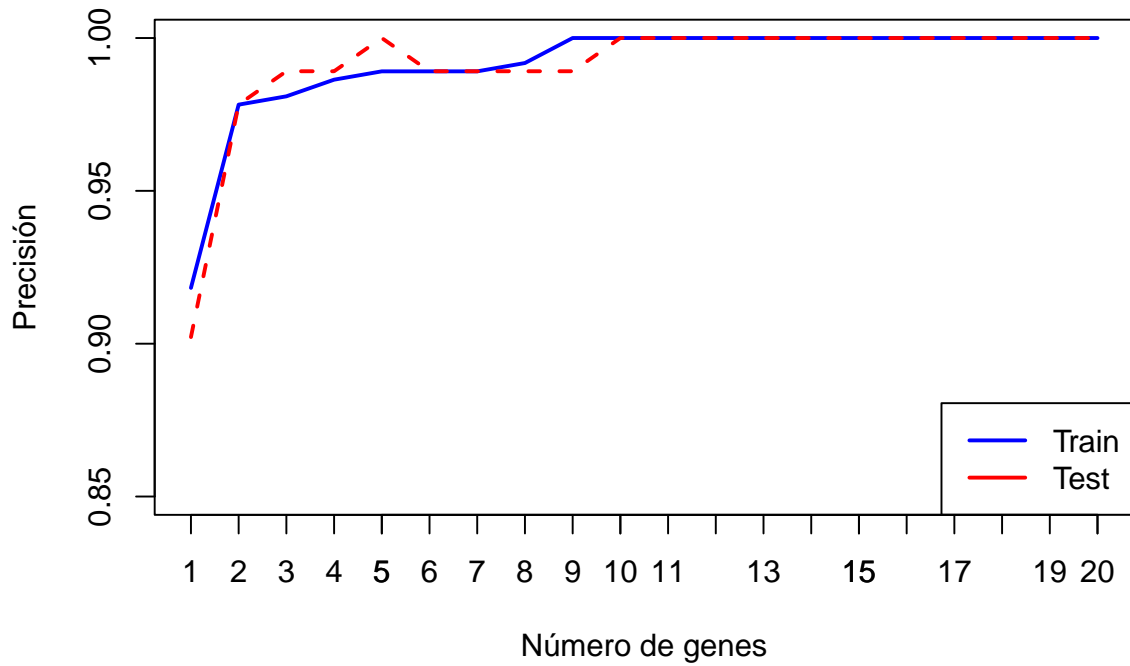
  pred_test <- predict(svm_mod, XTest[, vars_selected, drop = FALSE])
  confMatrixTest <- confusionMatrix(as.factor(pred_test), as.factor(YTest))
  ACCTestRF[particion,i] <- confMatrixTest$overall["Accuracy"]

}

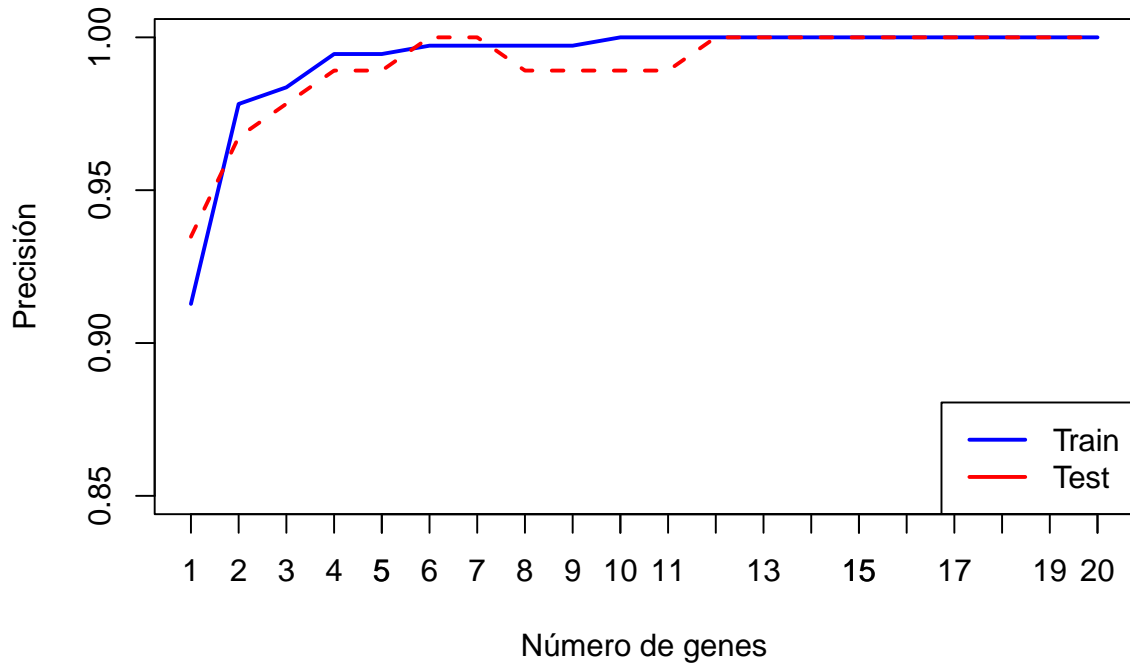
plot(1:nVarsMAX, ACCTrnRF[particion, ], type = "l", col = "blue", lwd = 2,
  ylim = c(0.85, 1), xlab = "Número de genes", ylab = "Precisión",
  main = paste("Partición", particion, ": SVM con Random Forest"))
lines(1:nVarsMAX, ACCTestRF[particion, ], col = "red", lwd = 2, lty = 2)
legend("bottomright", legend = c("Train", "Test"), col = c("blue", "red"),
  lty = 1, lwd = 2)
axis(1, at = 1:nVarsMAX)
}

```

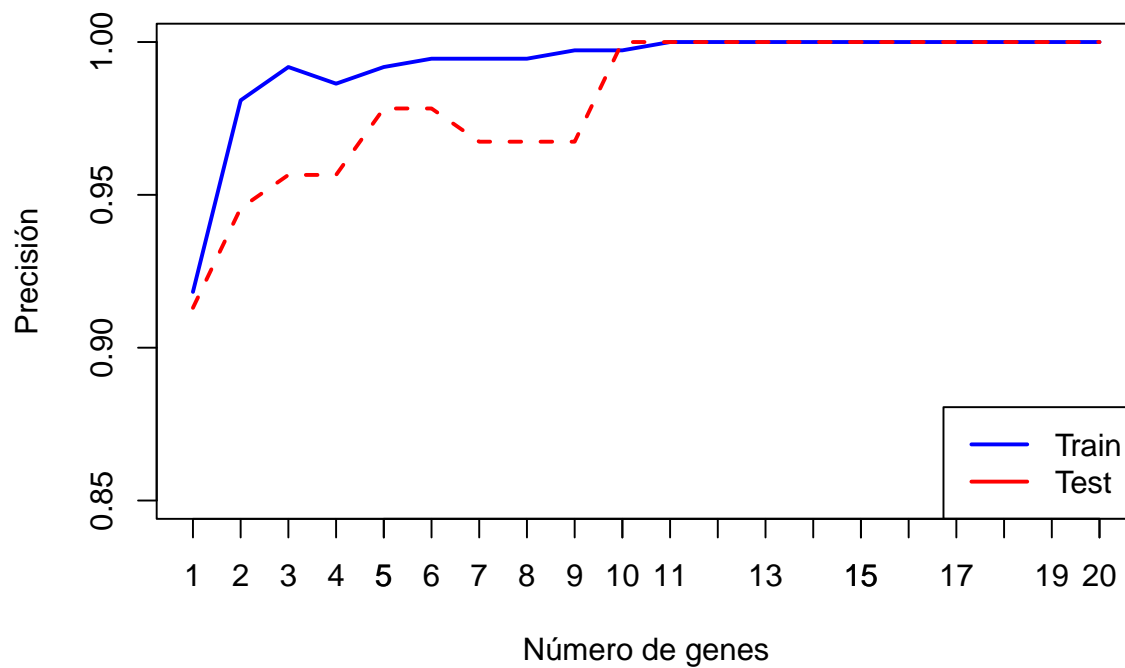
Partición 1 : SVM con Random Forest



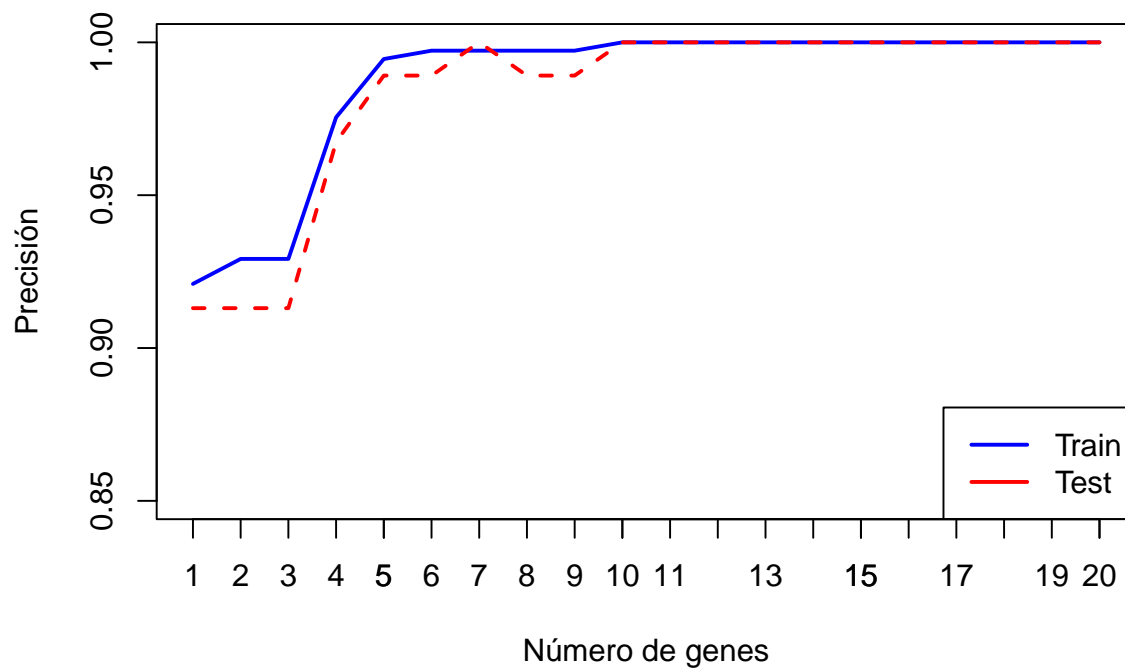
Partición 2 : SVM con Random Forest



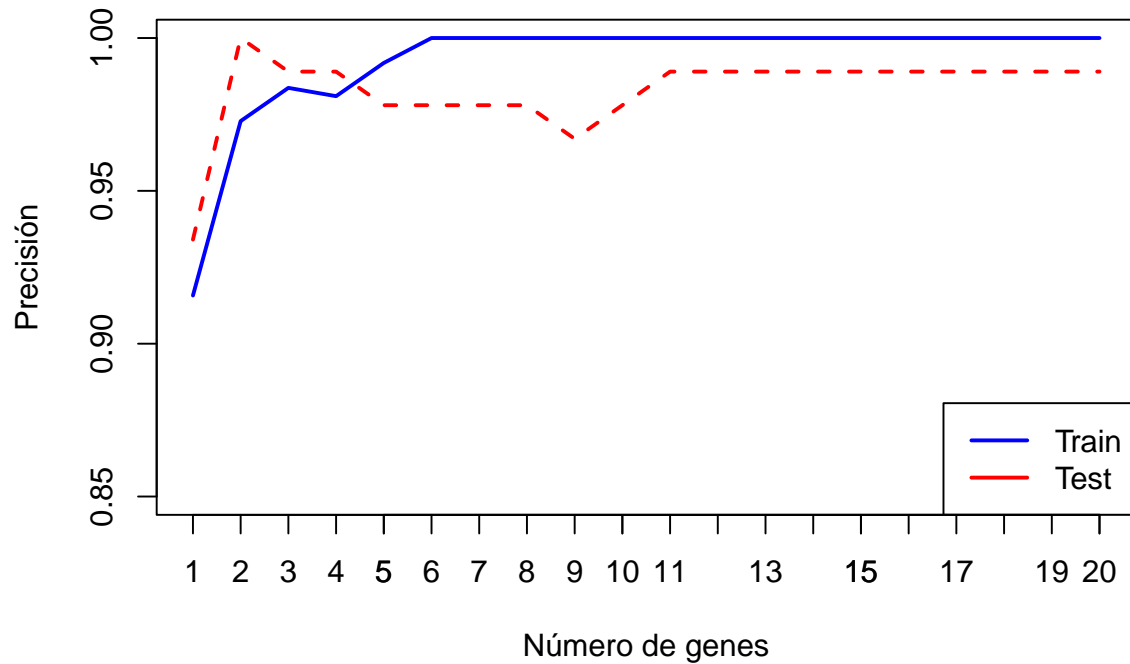
Partición 3 : SVM con Random Forest



Partición 4 : SVM con Random Forest

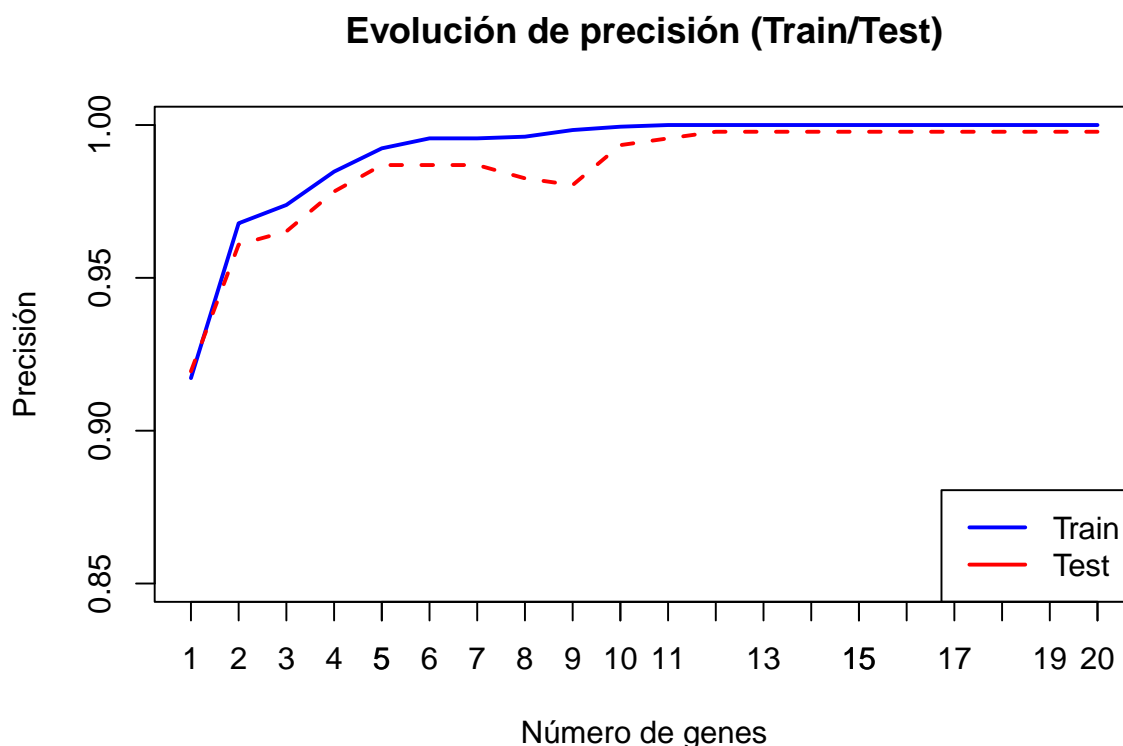


Partición 5 : SVM con Random Forest



```
meanACCTrnRF <- colMeans(ACCTrnRF)
meanACCTestRF <- colMeans(ACCTestRF)

plot(1:20, meanACCTrnRF[1:20], type = "l", col = "blue", lwd = 2,
     ylim = c(0.85, 1),
     xlab = "Número de genes", ylab = "Precisión",
     main = "Evolución de precisión (Train/Test)", xlim = c(1, 20))
lines(1:20, meanACCTestRF[1:20], col = "red", lwd = 2, lty = 2)
legend("bottomright", legend = c("Train", "Test"), col =
      c("blue", "red"), lty = 1, lwd = 2)
axis(1, at = 1:20)
```

A continuación, tras ver este código, vamos a explicar cada una de las secciones junto con sus resultados. Como ya se ha comentado, se ha escogido el algoritmo de clasificación **SVM** y el algoritmo de selección de características **Random Forest**.

Este código realiza una validación cruzada de 5 particiones (5-fold CV) para evaluar el rendimiento de un modelo SVM, utilizando un ranking de genes mediante Random Forest. El objetivo principal es analizar cómo varía la precisión del modelo al utilizar diferentes cantidades de genes seleccionados como variables predictoras.

En primer lugar, se establecen los parámetros necesarios, como el número máximo de genes a evaluar (nVarsMAX, que se ha cogido 20 al igual que en clase) y el número de particiones para la validación cruzada (nfolds, que deben de ser 5). A continuación, se inicializan matrices para almacenar los resultados de precisión tanto en el conjunto de entrenamiento como en el conjunto de prueba, así como las predicciones generadas por el modelo. Se hace una nueva selección de características en cada iteración para luego obtener un ranking diferente en cada una de ellas utilizando el algoritmo Random Forest.

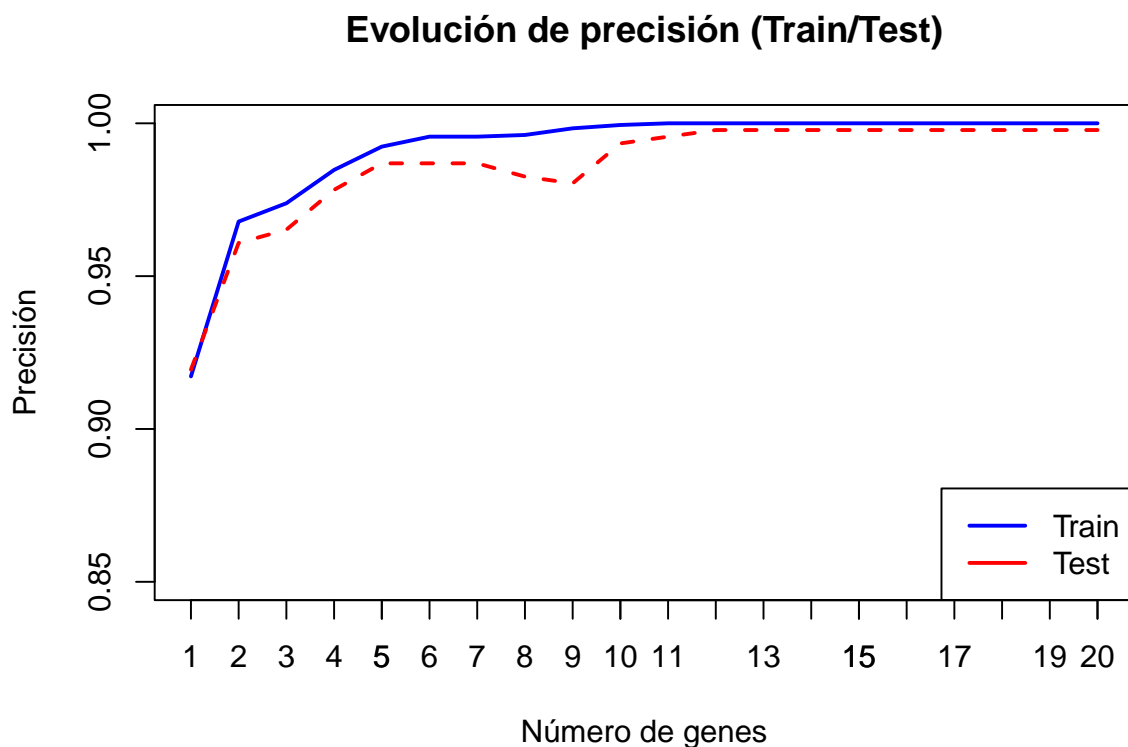
El código luego divide los datos en 5 particiones estratificadas usando la función `cvGenStratified`, y dentro de un bucle para cada partición, separa los datos en conjuntos de entrenamiento y prueba. Para cada número de características seleccionadas desde el ranking, entrena un modelo SVM con un kernel radial. La precisión de la predicción del modelo se evalúa tanto en el conjunto de entrenamiento como en el de prueba, y los resultados se almacenan en matrices específicas. Las predicciones del modelo para cada partición de prueba también se guardan en una matriz para su posterior análisis. Además de mostrar las gráficas de cada partición para poder ver la evolución entre una y otra.

Una vez que el proceso de validación cruzada ha finalizado, se calculan las medias de precisión de entrenamiento y test según el número de genes seleccionados (`colMeans`), para poder mostrar luego su evolución de la precisión de manera gráfica dependiendo del genes utilizados. Esta gráfica compara el rendimiento en los conjuntos de entrenamiento y test, y permite identificar cuántos genes ofrecen el mejor equilibrio entre precisión y complejidad del modelo.

En primer lugar, se mostrará una gráfica donde se vea la evolución en el número de genes para los valores medios de las 5CV en Trn y en Test. Como se ha comentado anteriormente se han ido guardando esos resultados en cada iteración, permitiendo ahora hacer la media de la Accuracy de train y test.

```
meanACCTrnRF <- colMeans(ACCTrnRF)
meanACCTestRF <- colMeans(ACCTestRF)

plot(1:20, meanACCTrnRF[1:20], type = "l", col = "blue", lwd = 2,
     ylim = c(0.85, 1),
     xlab = "Número de genes", ylab = "Precisión",
     main = "Evolución de precisión (Train/Test)", xlim = c(1, 20))
lines(1:20, meanACCTestRF[1:20], col = "red", lwd = 2, lty = 2)
legend("bottomright", legend = c("Train", "Test"), col =
      c("blue", "red"), lty = 1, lwd = 2)
axis(1, at = 1:20)
```



En esta gráfica se puede observar que en su mayoría la línea del *Train* se encuentra levemente por encima

de *Test*. Se observa que con dos genes hay un pico y podría ser una buena elección de huella (2 genes) ya que se encuentra aproximadamente en el 97%. Ocurre lo mismo con el número de genes igual a 4, 5 y 10 genes, la elección del número de genes dependerá de si queremos un modelo más simple o robusto.

En este caso, la mejor opción sería quedarnos con una huella de entre 4 y 5 genes. Creo que la mejor opción sería elegir **4 genes para la huella** ya que sigue siendo un número pequeño de genes y proporciona buena precisión.

Lo siguiente que vamos a realizar es mostrar los mejores genes seleccionados por Random Forest y verificar si este ranking coincide a través de las diferentes particiones.

```
for (i in 1:nfolds) {
  cat("\nPartición", i, ":\n")
  print(rankingSRF[i, ])
}
```

```
##
## Partición 1 :
## [1] "APLN"      "LINP1"     "LHX6"      "CHST4"     "AC022211.2"
## [6] "ESM1"      "DPP10"     "L1CAM"     "ARSF"      "AP000866.1"
## [11] "SMIM24"    "KRT19"     "QRFPR"     "ANXA8"     "SPINK1"
## [16] "AC104117.4" "COL22A1"   "TAT"       "CERKL"     "VWDE"
##
## Partición 2 :
## [1] "APLN"      "LINP1"     "LHX6"      "KRT19"     "ESM1"
## [6] "AP000866.1" "CHST4"     "AC022211.2" "DPP10"     "ARSF"
## [11] "L1CAM"     "QRFPR"     "SMIM24"    "CD46"      "GACAT2"
## [16] "S100A5"    "ANXA8"     "F2"        "USH1C"     "LIX1L.AS1"
##
## Partición 3 :
## [1] "APLN"      "LINP1"     "LHX6"      "AC022211.2" "CHST4"
## [6] "ESM1"      "DPP10"     "L1CAM"     "AP000866.1" "KRT19"
## [11] "ARSF"      "CD46"      "SCAMP3"    "ANXA8"     "CAPN10.DT"
## [16] "CSMD1"     "CHMP1B2P"  "LIX1L.AS1" "S100A5"     "SLC38A4"
##
## Partición 4 :
## [1] "APLN"      "AC022211.2" "LHX6"      "LINP1"     "AP000866.1"
## [6] "CHST4"     "L1CAM"      "ESM1"      "DPP10"     "ARSF"
## [11] "KRT19"     "CAPN10.DT"  "ITIH1"     "SCAMP3"    "CD46"
## [16] "ANXA8"     "SMIM24"     "F2"        "SPINK1"     "VWDE"
##
```

Partición 5 :

```
## [1] "APLN"      "LINP1"      "LHX6"      "AC022211.2" "CHST4"
## [6] "APO00866.1" "KRT19"      "ESM1"      "ARSF"      "DPP10"
## [11] "F2"        "L1CAM"      "SCAMP3"     "S100A5"     "CAPN10.DT"
## [16] "ANXA8"     "KLHL20"     "AC026316.3" "SMIM24"     "QRFPR"
```

Se puede observar como varían el orden de los genes según la partición correspondiente. A continuación vamos a continuar el estudio para ver que número de genes será elegido para la huella y luego elijeremos la huella.

La siguiente comprobación que vamos a hacer, será comprobar las métricas que han habido en cada partición:

ACCTrnRF

```
##          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.9182561 0.9782016 0.9809264 0.9863760 0.9891008 0.9891008 0.9891008
## [2,] 0.9128065 0.9782016 0.9836512 0.9945504 0.9945504 0.9972752 0.9972752
## [3,] 0.9182561 0.9809264 0.9918256 0.9863760 0.9918256 0.9945504 0.9945504
## [4,] 0.9209809 0.9291553 0.9291553 0.9754768 0.9945504 0.9972752 0.9972752
## [5,] 0.9157609 0.9728261 0.9836957 0.9809783 0.9918478 1.0000000 1.0000000
##          [,8]      [,9]      [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17]
## [1,] 0.9918256 1.0000000 1.0000000      1      1      1      1      1      1      1
## [2,] 0.9972752 0.9972752 1.0000000      1      1      1      1      1      1      1
## [3,] 0.9945504 0.9972752 0.9972752      1      1      1      1      1      1      1
## [4,] 0.9972752 0.9972752 1.0000000      1      1      1      1      1      1      1
## [5,] 1.0000000 1.0000000 1.0000000      1      1      1      1      1      1      1
##          [,18] [,19] [,20]
## [1,]      1      1      1
## [2,]      1      1      1
## [3,]      1      1      1
## [4,]      1      1      1
## [5,]      1      1      1
```

A medida que se aumenta el número de genes, las métricas de precisión también mejoran. Esto se muestra en las primeras columnas (de 1 a 10 genes), donde la precisión comienza entre 0.91 y 0.92, y aumenta progresivamente hasta alcanzar valores cercanos a 1. A partir de la sexta columna (6 genes), la precisión empieza a alcanzar en alguna partición 1.0000. Esto indica que, con al menos 6 genes, el modelo es capaz de clasificar correctamente todos los datos de entrenamiento sin errores. Las cinco particiones tienen valores muy similares en cada etapa, lo que sugiere que el modelo se comporta de manera consistente independientemente de la división de los datos.

Se observa que con 10 genes las precisiones son prácticamente perfectas, lo que indica que el modelo es altamente robusto con este número de genes. Sin embargo, la selección del número final de genes dependerá de los objetivos del análisis y el balance entre simplicidad y robustez del modelo. Por ejemplo, con 4 genes ya se obtienen precisiones bastante buenas, aunque levemente inferiores a las que se obtienen con 6 genes. Lo mismo ocurre con 8 genes, aunque este número implica una mayor complejidad, sin un aumento significativo en las métricas respecto a los 6 genes.

Por estas razones, he decidido utilizar **4 genes** en mi estudio. Este número permite alcanzar una buena precisión mientras se mantiene un modelo simple y eficiente. Aunque cualquiera de los valores planteados habría sido una elección válida. Este análisis me ha permitido confirmar que la elección que hice anteriormente es correcta, también coincide con el número elegido en el apartado **3.3.2**.

Esto ocurre para los datos de TRN. Ahora comprobaremos los de TEST.

ACCTestRF							
##	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
## [1,]	0.9021739	0.9782609	0.9891304	0.9891304	1.0000000	0.9891304	0.9891304
## [2,]	0.9347826	0.9673913	0.9782609	0.9891304	0.9891304	1.0000000	1.0000000
## [3,]	0.9130435	0.9456522	0.9565217	0.9565217	0.9782609	0.9782609	0.9673913
## [4,]	0.9130435	0.9130435	0.9130435	0.9673913	0.9891304	0.9891304	1.0000000
## [5,]	0.9340659	1.0000000	0.9890110	0.9890110	0.9780220	0.9780220	0.9780220
##	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]	[,14]
## [1,]	0.9891304	0.9891304	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
## [2,]	0.9891304	0.9891304	0.9891304	0.9891304	1.0000000	1.0000000	1.0000000
## [3,]	0.9673913	0.9673913	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
## [4,]	0.9891304	0.9891304	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
## [5,]	0.9780220	0.9670330	0.9780220	0.9890110	0.9890111	0.9890111	0.9890111
##	[,15]	[,16]	[,17]	[,18]	[,19]	[,20]	
## [1,]	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	
## [2,]	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	
## [3,]	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	
## [4,]	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	
## [5,]	0.9890111	0.9890111	0.9890111	0.9890111	0.9890111	0.9890111	

Los resultados obtenidos para las métricas de precisión en el conjunto de datos TEST muestran un comportamiento similar al de los datos de TRN, con una tendencia de mejora a medida que aumentamos el número de genes. En general, la precisión se incrementa y alcanza valores cercanos a 1.0000, lo que indica que el modelo clasifica correctamente los datos en la mayoría de las particiones. Al igual que con los datos de entrenamiento, la precisión comienza en valores alrededor de 0.91 para un gen, y aumenta hasta valores cercanos a 1.0000 a medida que aumentamos el número de genes (10 genes y más). Esto

indica que el modelo sigue mejorando a medida que se incorporan más genes, y que con 10 genes o más, el modelo es capaz de clasificar correctamente todos los datos de prueba sin errores.

A pesar de que 10 genes logran la precisión perfecta, el modelo ya muestra un rendimiento muy sólido a partir de 4 genes, con una precisión de 0.9594595 o 0.9891304 en algunas particiones. Esto refuerza mi decisión de usar **4 genes** como una opción eficiente y equilibrada para el modelo. Los resultados en el conjunto de TEST son coherentes con los obtenidos en el conjunto de TRN, indicando que el modelo tiene un buen poder de generalización y es capaz de predecir correctamente en datos no vistos. En cuanto a la selección de genes, se confirma que 4 genes proporcionan un gran rendimiento sin la necesidad de una mayor complejidad, aunque 10 genes también muestran una precisión perfecta, al igual que 8, 6 o 5.

Ahora he decidido mostrar las medias de las precisiones tanto de TRN como TEST para confirmar que coinciden con lo observado en la tabla.

```
meanACCTrnRF
```

```
## [1] 0.9172121 0.9678622 0.9738508 0.9847515 0.9923750 0.9956403 0.9956403
## [8] 0.9961853 0.9983651 0.9994550 1.0000000 1.0000000 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
```

```
meanACCTestRF
```

```
## [1] 0.9194219 0.9608696 0.9651935 0.9782370 0.9869087 0.9869087 0.9869087
## [8] 0.9825609 0.9803631 0.9934305 0.9956283 0.9978022 0.9978022 0.9978022
## [15] 0.9978022 0.9978022 0.9978022 0.9978022 0.9978022 0.9978022
```

Tras analizar estos resultados, considero acertada la elección de utilizar **4 genes como huella**, ya que la precisión media obtenida en los datos de entrenamiento es 0.9847515 y en los datos de prueba es 0.9782370. Estos valores son bastante buenos y reflejan un rendimiento excelente tanto en los datos de entrenamiento como en los de prueba. Además, se observa que al aumentar el número de genes, las medias se mantienen estables, lo que indica que añadir más genes no aporta mejoras significativas al rendimiento del modelo. Por lo tanto, con 6 genes se logra un buen balance entre rendimiento y eficiencia, y esta elección resulta ser la más adecuada para el modelo.

4.1 Resultado de la huella final y clasificador en 5CV

Como se mencionó anteriormente, utilizaremos **4 genes como huella**. En este apartado, llevaremos a cabo una validación cruzada con 5 particiones (5CV) para evaluar las métricas del modelo, generar la matriz de confusión y determinar los genes específicos que conformarán la huella definitiva. El modelo SVM será entrenado con los 4 genes seleccionados, tanto en los conjuntos de entrenamiento como de prueba.

Adicionalmente, almacenaremos todos los genes en una variable separada para facilitar la visualización y la selección final de genes en etapas posteriores.

```
set.seed(7)
nfolds <- 5 # Número de particiones para 5CV
num_genes <- 4 # Número de genes que he elegido
foldIDx <- cvGenStratified(LABELS, nfolds)

# Inicialización de matrices y listas necesarias
ACCTrnRF_final <- matrix(0, nfolds, 1)
ACCTestRF_final <- matrix(0, nfolds, 1)
cmTestFinal <- matrix(0, nrow=length(unique(LABELS)), ncol=length(unique(LABELS)))

# Lista para guardar matrices de confusión de cada partición
confusionMatricesByPartition <- vector("list", nfolds)

# Inicializar rankingSRF para almacenar todos los genes en cada partición
rankingSRF <- vector("list", nfolds)

# Bucle para las particiones
for (particion in seq(1, nfolds)) {
  cat("Partición", particion, "\n")

  # Índices de Train y Test
  indexTest <- which(foldIDx == particion)
  indexTrn <- setdiff(seq(1, length(LABELS)), indexTest)

  # Dividir datos en entrenamiento y test
  XTrn <- as.data.frame(t(MATRIZ[,indexTrn]))
  YTrn <- as.factor(LABELS[indexTrn])
  XTest <- as.data.frame(t(MATRIZ[,indexTest]))
  YTest <- as.factor(LABELS[indexTest])

  # Calcular el ranking de características para todos los genes
  rankingRF <- featureSelection(XTrn, YTrn, mode = "rf", vars_selected =
                                rownames(DEGsMatrix))

  # Guardar el ranking completo de genes seleccionados para esta partición
  rankingSRF[[particion]] <- rankingRF
}
```

```

# Seleccionar los primeros 4 genes para entrenar el modelo
selectedGenes <- rankingRF[1:num_genes]
cmPartition <- matrix(0, nrow = length(unique(YTest)), ncol =
                      length(unique(YTest)))

# Modelo SVM Train con los 4 genes seleccionados
svm_mod <- svm(XTrn[, selectedGenes, drop = FALSE], as.factor(YTrn),
              kernel = "radial")
pred_train <- predict(svm_mod, XTrn[, selectedGenes, drop = FALSE])
confMatrixTrn <- confusionMatrix(as.factor(pred_train), as.factor(YTrn))
ACCTrnRF_final[particion] <- confMatrixTrn$overall["Accuracy"]

# Modelo SVM Test con los 4 genes seleccionados
pred_test <- predict(svm_mod, XTest[, selectedGenes, drop = FALSE])
confMatrixTest <- confusionMatrix(as.factor(pred_test), as.factor(YTest))
ACCTestRF_final[particion] <- confMatrixTest$overall["Accuracy"]

# Guardar la matriz de confusión de esta partición
cmPartition <- confMatrixTest$table
confusionMatricesByPartition[[particion]] <- cmPartition

# Acumular la matriz de confusión para la matriz final
cmTestFinal <- cmTestFinal + confMatrixTest$table
}

# Calcular medias de 5CV para la huella final (4 genes)
meanACCTrnRF_final <- mean(ACCTrnRF_final)
meanACCTestRF_final <- mean(ACCTestRF_final)

```

Ahora se muestran las medias totales del entrenamiento y test.

```
meanACCTrnRF_final
```

```
## [1] 0.9847515
```

```
meanACCTestRF_final
```

```
## [1] 0.978237
```

A continuación, se mostrarán las matrices de confusión de cada partición:


```

for (particion in seq_along(confusionMatricesByPartition)) {
  cat(paste("Partición", particion, ":\n"))
  print(confusionMatricesByPartition[[particion]])
}

```

```

## Partición 1 :
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      6      0      0
##   Healthy    0     11      0
##   LIHC       1      0     74
## Partición 2 :
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      6      0      1
##   Healthy    0     12      0
##   LIHC       0      0     73
## Partición 3 :
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      2      0      0
##   Healthy    0     12      0
##   LIHC       4      0     74
## Partición 4 :
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      5      0      0
##   Healthy    0     10      0
##   LIHC       1      2     74
## Partición 5 :
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      6      0      0
##   Healthy    0     10      0
##   LIHC       0      1     74

```

Y la matriz de confusión acumulada:

```
cmTestFinal
```

```
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      25      0    1
##   Healthy    0     55    0
##   LIHC       6      3  369
```

Ahora comprobamos que la matriz de confusión acumulada es correcta, mostrando el número de instancias por cada clase.

```
table(LABELS)
```

```
## LABELS
##   CHOL Healthy   LIHC
##    31      58    370
```

La matriz de confusión indica que el modelo tiene un buen desempeño en la clasificación de las tres clases: CHOL, Healthy y LIHC. La mayoría de las instancias están clasificadas correctamente, con 25 casos de CHOL, 55 de Healthy y 369 de LIHC correctamente identificados. Sin embargo, hay algunos errores de clasificación. Por ejemplo, 6 casos de LIHC fueron clasificados como CHOL, 3 casos de LIHC fueron clasificados como Healthy, y 1 caso de CHOL fue clasificado como LIHC. A pesar de estos errores, la cantidad de aciertos sigue siendo alta, y los errores son relativamente pocos.

A continuación, se mostrarán un ranking de los 10 primeros genes de cada una de las particiones, lo que nos permitirá escoger los 4 genes correspondientes para nuestra huella:

```
for (particion in seq(1, nfolds)) {
  cat("Partición", particion, "\n")
  cat(rankingSRF[[particion]][1:10], "\n\n")
}
```

```
## Partición 1 :
## APLN LINP1 LHX6 CHST4 AC022211.2 ESM1 DPP10 L1CAM ARSF AP000866.1
##
## Partición 2 :
## APLN LINP1 LHX6 KRT19 ESM1 AP000866.1 CHST4 AC022211.2 DPP10 ARSF
##
## Partición 3 :
## APLN LINP1 LHX6 AC022211.2 CHST4 ESM1 DPP10 L1CAM AP000866.1 KRT19
##
```

```
## Partición 4 :
## APLN AC022211.2 LHX6 LINP1 AP000866.1 CHST4 L1CAM ESM1 DPP10 ARSF
##
## Partición 5 :
## APLN LINP1 LHX6 AC022211.2 CHST4 AP000866.1 KRT19 ESM1 ARSF DPP10
```

Observando los genes, nos quedaremos con los siguientes 4 genes: **APLN**, **LINP1**, **LHX6** y **AC022211.2**. Estos genes son los más consistentes, ya que aparecen en todas las particiones. Random Forest genera un *ranking de características basado en su importancia para la predicción*, y los genes seleccionados son aquellos que tienen un mayor impacto en el modelo. Los 4 genes elegidos son los más frecuentes e importantes en el ranking, ya que son los que aparecen con mayor regularidad a través de las particiones y, por lo tanto, se consideran los más relevantes.

Ahora se mostrará un boxplot de los genes seleccionados:

```
# Definir los genes de interés
genes_seleccionados <- c("APLN", "LINP1", "LHX6", "AC022211.2")

# Transponer la matriz si los genes están en las filas
matriz_df <- as.data.frame(t(MATRIZ))

# Convertir los nombres de fila en una columna de muestras
matriz_df$Sample_ID <- rownames(matriz_df)

# Asegurar que los genes están como columnas
colnames(matriz_df) <- c(rownames(MATRIZ), "Sample_ID")

# Filtrar solo los genes seleccionados
genes_existentes <- genes_seleccionados[genes_seleccionados %in% colnames(matriz_df)]
matriz_filtrada <- matriz_df[, c("Sample_ID", genes_existentes), drop = FALSE]

# Crear las etiquetas de clase
LABELS2 <- c(rep("CHOL", 31), rep("Healthy", 58), rep("LIHC", 370))

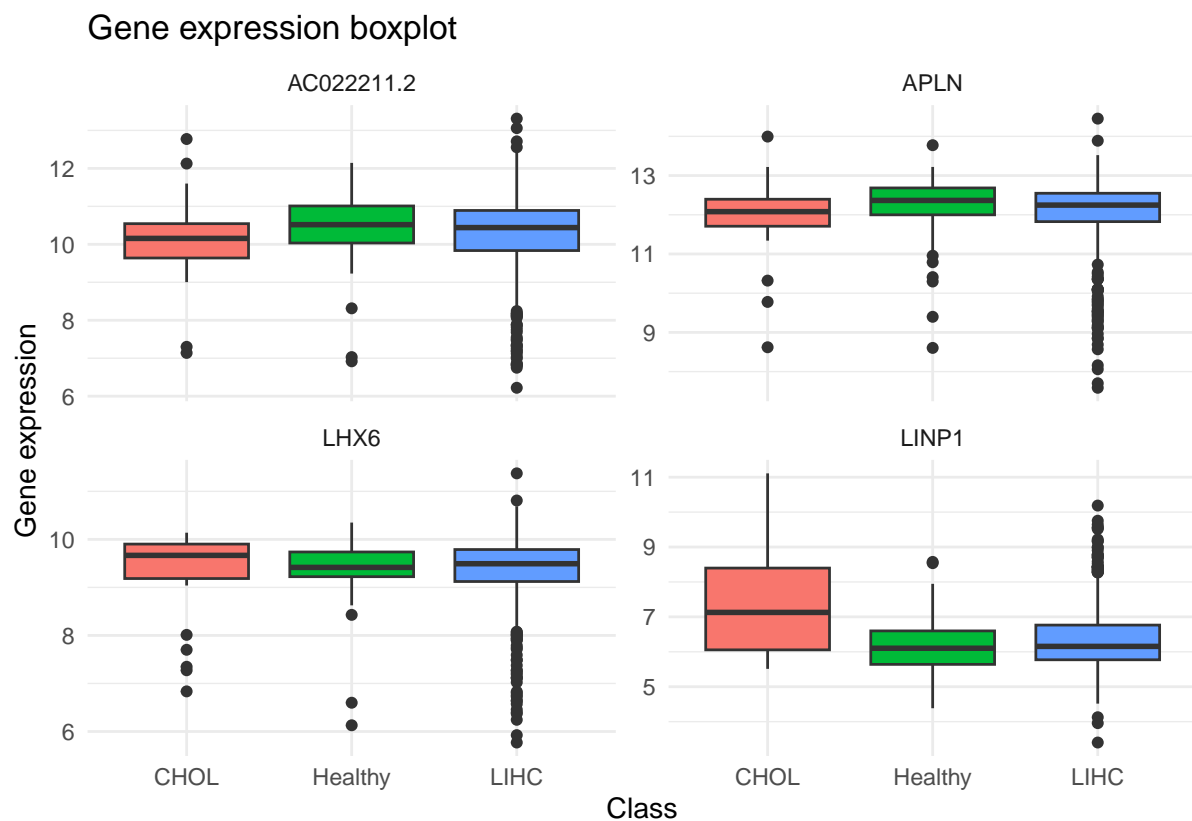
# Verificar que la cantidad de etiquetas coincide con las muestras
if (length(LABELS2) != nrow(matriz_filtrada)) {
  stop("El número de etiquetas no coincide con el número de muestras.")
}

# Añadir las etiquetas al dataframe
```

```
matriz_filtrada$Label <- factor(LABELS2)

# Convertir los datos a formato largo (long format) para ggplot2
data_long <- matriz_filtrada %>%
  pivot_longer(cols = -c(Sample_ID, Label), names_to = "Gen", values_to = "Expresión")

# Generar los boxplots para cada gen
ggplot(data_long, aes(x = Label, y = Expresión, fill = Label)) +
  geom_boxplot() +
  facet_wrap(~Gen, scales = "free_y") +
  labs(title = "Gene expression boxplot",
       x = "Class",
       y = "Gene expression") +
  theme_minimal() +
  theme(legend.position = "none")
```



5 Resultado de la huella final escogida

Para obtener el resultado de la huella final con los 4 genes seleccionados (**APLN**, **LINP1**, **LHX6** y **AC022211.2**) y entrenar un clasificador en validación cruzada (5-CV), además de calcular la matriz de confusión , se deberían seguir los siguientes pasos.

- Dividir los datos en dos conjuntos: uno de entrenamiento (80%) y otro de test (20%).
- Entrenar un modelo SVM optimizando sus parámetros mediante un GridSearch.
- Se realizarán predicciones sobre el conjunto de prueba y se evaluará el rendimiento del modelo a través de una matriz de confusión.

```
set.seed(7) # Semilla propia
nfolds <- 5 # Número de particiones para 5CV

# Definir los rangos para los parámetros de SVM
grid <- expand.grid(sigma = c(0,0.01, 0.02, 0.025, 0.03, 0.04,0.05, 0.06, 0.07,
  0.08, 0.09, 0.1, 0.25, 0.5, 0.75,0.9),
  C = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5, 2,5))

# Selección de genes finales
finalGenes <- c("APLN", "LINP1", "LHX6", "AC022211.2")

# Etiquetas convertidas a factor
trueLabels <- factor(LABELS)

# Filtrar los datos para incluir solo los genes seleccionados
X_selected <- t(MATRIZ)[, finalGenes]
Y <- trueLabels # Etiquetas como factor

# Crear índices para la validación cruzada
foldIdx <- createFolds(Y, k = nfolds, list = TRUE)

# Inicializar matriz acumulativa de confusión
cmTestFinal <- matrix(0, nrow = length(unique(Y)), ncol = length(unique(Y)))

# Lista para guardar matrices de confusión de cada partición
confusionMatricesByPartition <- vector("list", nfolds)
```

```

# Bucle para las particiones
for (particion in seq_len(nfolds)) {
  cat("Partición", particion, "\n")

  # Índices de Train y Test
  indexTest <- foldIDx[[particion]]
  indexTrn <- setdiff(seq_len(length(Y)), indexTest)

  # Dividir datos en entrenamiento y test
  X_train <- X_selected[indexTrn, ]
  Y_train <- Y[indexTrn]
  X_test <- X_selected[indexTest, ]
  Y_test <- Y[indexTest]

  # Entrenar el modelo con validación cruzada
  train_control <- trainControl(method = "cv", number = 5)
  svm_model <- train(x = X_train, y = Y_train, method = "svmRadial",
                    trControl = train_control, tuneGrid = grid)

  # Mejores hiperparámetros encontrados
  best_params <- svm_model$bestTune

  # Entrenar SVM con los mejores hiperparámetros
  final_model <- svm(x = X_train, y = Y_train, kernel = "radial",
                    C = best_params$C, sigma = best_params$sigma, probability = TRUE)

  # Predicciones en Test
  pred_test <- predict(final_model, X_test, probability = TRUE)

  # Matriz de confusión
  conf_matrix <- confusionMatrix(pred_test, Y_test)
  confusionMatricesByPartition[[particion]] <- conf_matrix$table

  # Acumular la matriz de confusión final
  cmTestFinal <- cmTestFinal + conf_matrix$table
}

## Partición 1
## Partición 2

```

```
## Partición 3
## Partición 4
## Partición 5
```

```
# Mostrar matriz de confusión final acumulada
```

```
cmTestFinal
```

```
##           Reference
## Prediction CHOL Healthy LIHC
##    CHOL      24      0    1
##    Healthy    0     57    0
##    LIHC       7      1  369
```

```
# Sumar todas las matrices de confusión para obtener la final
```

```
cmTestFinal <- Reduce("+", confusionMatricesByPartition)
```

```
# Mostrar la matriz de confusión final acumulada
```

```
cat("Matriz de confusión global acumulada:\n")
```

```
## Matriz de confusión global acumulada:
```

```
print(cmTestFinal)
```

```
##           Reference
## Prediction CHOL Healthy LIHC
##    CHOL      24      0    1
##    Healthy    0     57    0
##    LIHC       7      1  369
```

```
# Extraer valores de la matriz de confusión
```

```
TP <- diag(cmTestFinal) # Verdaderos Positivos
```

```
FP <- colSums(cmTestFinal) - TP # Falsos Positivos
```

```
FN <- rowSums(cmTestFinal) - TP # Falsos Negativos
```

```
TN <- sum(cmTestFinal) - (TP + FP + FN) # Verdaderos Negativos
```

```
# Calcular métricas globales por clase
```

```
accuracy_global <- sum(TP) / sum(cmTestFinal) # Precisión global
```

```
sensitivity_global <- TP / (TP + FN) # (Recall)
```

```
specificity_global <- TN / (TN + FP) # Especificidad
```

```
f1_score_global <- 2 * TP / (2 * TP + FP + FN) # F1-Score
```

```
balanced_accuracy_global <- (sensitivity_global + specificity_global) / 2
```

```
# Crear un dataframe con los resultados por clase
```

```
metrics_df <- data.frame(  
  Class = levels(Y),  
  Sensitivity = round(sensitivity_global, 4),  
  Specificity = round(specificity_global, 4),  
  F1_Score = round(f1_score_global, 4),  
  Balanced_Accuracy = round(balanced_accuracy_global, 4)  
)
```

```
# Mostrar métricas generales
```

```
cat("\nPrecisión Global:", round(accuracy_global, 4), "\n")
```

```
##
```

```
## Precisión Global: 0.9804
```

```
cat("\nMétricas por clase:\n")
```

```
##
```

```
## Métricas por clase:
```

```
print(metrics_df)
```

```
##           Class Sensitivity Specificity F1_Score Balanced_Accuracy  
## CHOL        CHOL      0.9600      0.9839   0.8571           0.9719  
## Healthy    Healthy    1.0000      0.9975   0.9913           0.9988  
## LIHC        LIHC      0.9788      0.9878   0.9880           0.9833
```

El modelo alcanza una precisión de 0.9804, lo que indica un excelente rendimiento en la clasificación. Al analizar las métricas por clase, se observa lo siguiente:

- **CHOL:** La métrica recall es del 96.00%, lo que significa que el modelo identifica correctamente la mayoría de los casos de esta clase. La especificidad es del 98.39%, lo que refleja una gran capacidad para identificar correctamente los casos negativos. La métrica F1_Score para esta clase es del 85.71%, indicando una relación favorable entre los verdaderos positivos y los falsos positivos.
- **Healthy:** El recall es del 100%, lo que indica que el modelo identifica correctamente todos los casos de la clase Healthy. La especificidad es del 99.75%, lo que significa que el modelo distingue correctamente los casos negativos. El F1_Score es del 99.13%, lo que refleja un excelente desempeño en la predicción de esta clase.

- **LIHC:** El recall es del 97.88%, lo que significa que el modelo clasifica correctamente la mayoría de los casos de LIHC. La especificidad es del 98.78%, lo que implica una gran capacidad para identificar correctamente los negativos. El F1_Score es del 98.80%, lo que indica un gran rendimiento al predecir esta clase.

El modelo muestra un buen desempeño global y por clase, con una alta capacidad para diferenciar entre las clases CHOL, Healthy y LIHC. Aunque hay algunos errores de clasificación en CHOL y LIHC, el modelo sigue mostrando un rendimiento sólido en general, con una precisión global destacada. Estos resultados reflejan la efectividad del modelo en este problema de clasificación multiclase.

6 Comparación con huella externa

En esta sección se presentarán los resultados obtenidos utilizando una huella externa, la cual fue generada a partir de un artículo que abordó el mismo problema que estamos investigando, el cáncer de hígado. El objetivo principal de este apartado es comparar los resultados de las distintas huellas y determinar cuál de ellas proporciona las mejores métricas.

La huella que se utilizará será la siguiente,: **MAPT**, **VIPR1**, **TNFRSF4**, **CCR1**, **BIRC5**, **RFX5**, **APLN**.

```
set.seed(7) # Semilla propia
nfolds <- 5 # Número de particiones para 5CV

# Definir los rangos para los parámetros de SVM
grid <- expand.grid(sigma = c(0,0.01, 0.02, 0.025, 0.03, 0.04,0.05, 0.06, 0.07,
  0.08, 0.09, 0.1, 0.25, 0.5, 0.75,0.9),
  C = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5, 2,5))

# Selección de genes finales
finalGenes <- c("MAPT", "VIPR1", "TNFRSF4", "CCR1", "BIRC5", "RFX5", "APLN")

# Etiquetas convertidas a factor
trueLabels <- factor(LABELS)

# Filtrar los datos para incluir solo los genes seleccionados
X_selected <- t(MATRIZ)[, finalGenes]
Y <- trueLabels # Etiquetas como factor

# Crear índices para la validación cruzada
foldIdx <- createFolds(Y, k = nfolds, list = TRUE)

# Inicializar matriz acumulativa de confusión
cmTestFinal <- matrix(0, nrow = length(unique(Y)), ncol = length(unique(Y)))

# Lista para guardar matrices de confusión de cada partición
confusionMatricesByPartition <- vector("list", nfolds)

# Bucle para las particiones
for (particion in seq_len(nfolds)) {
  cat("Partición", particion, "\n")
```

```

# Índices de Train y Test
indexTest <- foldIDx[[particion]]
indexTrn <- setdiff(seq_len(length(Y)), indexTest)

# Dividir datos en entrenamiento y test
X_train <- X_selected[indexTrn, ]
Y_train <- Y[indexTrn]
X_test <- X_selected[indexTest, ]
Y_test <- Y[indexTest]

# Entrenar el modelo con validación cruzada
train_control <- trainControl(method = "cv", number = 5)
svm_model <- train(x = X_train, y = Y_train,
                  method = "svmRadial", trControl = train_control, tuneGrid = grid)

# Mejores hiperparámetros encontrados
best_params <- svm_model$bestTune

# Entrenar SVM con los mejores hiperparámetros
final_model <- svm(x = X_train, y = Y_train, kernel = "radial",
                  C = best_params$C, sigma = best_params$sigma, probability = TRUE)

# Predicciones en Test
pred_test <- predict(final_model, X_test, probability = TRUE)

# Matriz de confusión
conf_matrix <- confusionMatrix(pred_test, Y_test)
confusionMatricesByPartition[[particion]] <- conf_matrix$table

# Acumular la matriz de confusión final
cmTestFinal <- cmTestFinal + conf_matrix$table
}

```

```

## Partición 1
## Partición 2
## Partición 3
## Partición 4
## Partición 5

```

```
# Mostrar matriz de confusión final acumulada
```

```
cmTestFinal
```

```
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      13      0    5
##   Healthy    0     53    2
##   LIHC      18      5  363
```

```
cmTestFinal <- Reduce("+", confusionMatricesByPartition)
```

```
cat("Matriz de confusión global acumulada:\n")
```

```
## Matriz de confusión global acumulada:
```

```
print(cmTestFinal)
```

```
##           Reference
## Prediction CHOL Healthy LIHC
##   CHOL      13      0    5
##   Healthy    0     53    2
##   LIHC      18      5  363
```

```
TP <- diag(cmTestFinal)
FP <- colSums(cmTestFinal) - TP
FN <- rowSums(cmTestFinal) - TP
TN <- sum(cmTestFinal) - (TP + FP + FN)
```

```
# Calcular métricas globales por clase
```

```
accuracy_global <- sum(TP) / sum(cmTestFinal)
sensitivity_global <- TP / (TP + FN)
specificity_global <- TN / (TN + FP)
f1_score_global <- 2 * TP / (2 * TP + FP + FN)
balanced_accuracy_global <- (sensitivity_global + specificity_global) / 2
```

```
# Crear un dataframe con los resultados por clase
```

```
metrics_df <- data.frame(
  Class = levels(Y),
  Sensitivity = round(sensitivity_global, 4),
```

```

Specificity = round(specificity_global, 4),
F1_Score = round(f1_score_global, 4),
Balanced_Accuracy = round(balanced_accuracy_global, 4)
)

# Mostrar métricas generales
cat("\nPrecisión Global:", round(accuracy_global, 4), "\n")

```

```

##
## Precisión Global: 0.9346

```

```

cat("\nMétricas por clase:\n")

```

```

##
## Métricas por clase:

```

```

print(metrics_df)

```

##	Class	Sensitivity	Specificity	F1_Score	Balanced_Accuracy
## CHOL	CHOL	0.7222	0.9592	0.5306	0.8407
## Healthy	Healthy	0.9636	0.9876	0.9381	0.9756
## LIHC	LIHC	0.9404	0.9041	0.9603	0.9223

El modelo muestra una precisión global de 0.9346, lo que refleja un buen rendimiento en la clasificación. La matriz de confusión indica que el modelo clasifica correctamente la mayoría de los casos, pero aún comete algunos errores. En particular, la clase CHOL presenta una sensibilidad del 72.22%, lo que significa que el modelo tiene dificultades para identificar todos los casos de esta clase. Sin embargo, la especificidad para CHOL es alta (95.92%), lo que muestra que el modelo es eficaz al clasificar correctamente los casos negativos.

Para la clase Healthy, el modelo alcanza una excelente sensibilidad de 96.36% y especificidad de 98.76%, lo que implica un alto rendimiento en la identificación de esta clase. El F1-Score de 0.9381 también refleja un buen equilibrio entre precisión y sensibilidad.

En cuanto a LIHC, el modelo obtiene una sensibilidad de 94.04% y un F1-Score de 0.9603, lo que indica una clasificación efectiva para esta clase. La especificidad de 90.41% también muestra una buena capacidad para evitar falsos positivos.

En resumen, el modelo tiene un buen desempeño en general, especialmente en Healthy y LIHC, pero podría mejorar en la clasificación de CHOL, donde la sensibilidad es más baja.

6.1 Conclusiones

El modelo con mi huella elegida supera al modelo con la huella externa en la mayoría de las métricas. La precisión global es considerablemente mayor con la huella elegida es de 0.9804 frente a 0.9346 de la huella externa, y los valores de sensibilidad, especificidad y F1-Score también son mejores, especialmente para las clases CHOL y LIHC. La huella externa muestra un desempeño más débil en CHOL, con una sensibilidad mucho menor, y también presenta dificultades en la especificidad de LIHC.

Esto sugiere que la huella elegida por mí ofrece un mejor rendimiento global y una mayor capacidad para diferenciar entre las clases, mientras que la huella externa muestra más errores, especialmente en la clasificación de CHOL y LIHC.

7 Enriquecimiento de genes para la huella seleccionada

En este apartado, se llevará a cabo un análisis de enriquecimiento para los genes seleccionados en la huella genética. Primero, se obtendrán los identificadores ENTREZ de los genes seleccionados. Luego, se realizará un análisis de Gene Ontology (GO) para identificar las categorías biológicas, moleculares y celulares más relevantes asociadas a estos genes, destacando aquellas con el mayor GeneRatio como enriquecidas. A continuación, se explorarán los pathways biológicos relacionados con los genes seleccionados, analizando su participación en diferentes rutas celulares y procesos biológicos. Finalmente, se investigarán las asociaciones con enfermedades utilizando los genes de la huella, evaluando las evidencias de su implicación en diversas condiciones patológicas.

7.1 Gene Ontology (GO)

```
# Genes seleccionados de la huella
genes_selected <- c("APLN", "LINP1", "LHX6", "AC022211.2")

# Cogemos los identificadores ENTREZ
entrezAnnotation <- getGenesAnnotation(genes_selected, attributes =
  c("external_gene_name", "entrezgene_id"), filter = "external_gene_name")

entrezGeneIds <- entrezAnnotation$entrezgene_id[!is.na(entrezAnnotation$entrezgene_id)]

# Se descarga informacion sobre los Gene Ontology
GOs <- geneOntologyEnrichment_updated(as.character(entrezGeneIds), geneType=
  "ENTREZ_GENE_ID")

save(GOs, file = "GOs")

load(file = "GOs")
```

Muestro los resultados de Gene Ontology:

GOs

##	ONTOLOGY	ID
##	G0:0021892	BP G0:0021892
##	G0:0042756	BP G0:0042756
##	G0:1903238	BP G0:1903238

## G0:0002031	BP G0:0002031
## G0:1904936	BP G0:1904936
## G0:0006044	BP G0:0006044
## G0:1903236	BP G0:1903236
## G0:0002029	BP G0:0002029
## G0:0022401	BP G0:0022401
## G0:0023058	BP G0:0023058
## G0:0097154	BP G0:0097154
## G0:0051923	BP G0:0051923
## G0:0021884	BP G0:0021884
## G0:0040037	BP G0:0040037
## G0:1901071	BP G0:1901071
## G0:1904996	BP G0:1904996
## G0:0021895	BP G0:0021895
## G0:1905564	BP G0:1905564
## G0:0002092	BP G0:0002092
## G0:0050901	BP G0:0050901
## G0:0021799	BP G0:0021799
## G0:0045823	BP G0:0045823
## G0:1903524	BP G0:1903524
## G0:0016266	BP G0:0016266
## G0:0040036	BP G0:0040036
## G0:1904994	BP G0:1904994
## G0:0002691	BP G0:0002691
## G0:1904706	BP G0:1904706
## G0:0006040	BP G0:0006040
## G0:0007595	BP G0:0007595
## G0:0021879	BP G0:0021879
## G0:0021795	BP G0:0021795
## G0:0060976	BP G0:0060976
## G0:0045776	BP G0:0045776
## G0:1905562	BP G0:1905562
## G0:0101023	BP G0:0101023
## G0:0021872	BP G0:0021872
## G0:0048260	BP G0:0048260
## G0:1902895	BP G0:1902895
## G0:0061756	BP G0:0061756
## G0:0022029	BP G0:0022029
## G0:0021885	BP G0:0021885

## G0:0045744	BP G0:0045744
## G0:2000630	BP G0:2000630
## G0:0048662	BP G0:0048662
## G0:0002090	BP G0:0002090
## G0:0045123	BP G0:0045123
## G0:1902893	BP G0:1902893
## G0:0061614	BP G0:0061614
## G0:0008543	BP G0:0008543
## G0:2000628	BP G0:2000628
## G0:0021954	BP G0:0021954
## G0:1904705	BP G0:1904705
## G0:0007589	BP G0:0007589
## G0:1990874	BP G0:1990874
## G0:0006493	BP G0:0006493
## G0:0001938	BP G0:0001938
## G0:0007631	BP G0:0007631
## G0:0010586	BP G0:0010586
## G0:0044344	BP G0:0044344
## G0:0048259	BP G0:0048259
## G0:0071774	BP G0:0071774
## G0:0021987	BP G0:0021987
## G0:0030879	BP G0:0030879
## G0:0090288	BP G0:0090288
## G0:0031623	BP G0:0031623
## G0:0045807	BP G0:0045807
## G0:0008277	BP G0:0008277
## G0:0048660	BP G0:0048660
## G0:0048659	BP G0:0048659
## G0:0001936	BP G0:0001936
## G0:0001764	BP G0:0001764
## G0:0008217	BP G0:0008217
## G0:0021543	BP G0:0021543
## G0:0001935	BP G0:0001935
## G0:0007369	BP G0:0007369
## G0:0021953	BP G0:0021953
## G0:0048469	BP G0:0048469
## G0:0008016	BP G0:0008016
## G0:0050679	BP G0:0050679
## G0:0006486	BP G0:0006486

## G0:0043413	BP G0:0043413
## G0:0002685	BP G0:0002685
## G0:0070085	BP G0:0070085
## G0:0033002	BP G0:0033002
## G0:0060047	BP G0:0060047
## G0:0003015	BP G0:0003015
## G0:1903522	BP G0:1903522
## G0:0006898	BP G0:0006898
## G0:0071695	BP G0:0071695
## G0:0021537	BP G0:0021537
## G0:1903039	BP G0:1903039
## G0:0030100	BP G0:0030100
## G0:0009101	BP G0:0009101
## G0:0022409	BP G0:0022409
## G0:0006790	BP G0:0006790
## G0:0021700	BP G0:0021700
## G0:0050878	BP G0:0050878
## G0:0090287	BP G0:0090287
## G0:0009100	BP G0:0009100
## G0:1903037	BP G0:1903037
## G0:0050900	BP G0:0050900
## G0:0050678	BP G0:0050678
## G0:0030900	BP G0:0030900
## G0:0007159	BP G0:0007159
## G0:0048732	BP G0:0048732
## G0:0050673	BP G0:0050673
## G0:0045785	BP G0:0045785
## G0:0005802	CC G0:0005802
## G0:0098791	CC G0:0098791
## G0:0071855	MF G0:0071855
## G0:0008146	MF G0:0008146
## G0:0016782	MF G0:0016782
## G0:0005179	MF G0:0005179
## G0:0001664	MF G0:0001664

##

Description

## G0:0021892	cerebral cortex GABAergic interneuron differentiation
## G0:0042756	drinking behavior
## G0:1903238	positive regulation of leukocyte tethering or rolling
## G0:0002031	G protein-coupled receptor internalization

G0:1904936 interneuron migration
 ## G0:0006044 N-acetylglucosamine metabolic process
 ## G0:1903236 regulation of leukocyte tethering or rolling
 ## G0:0002029 desensitization of G protein-coupled receptor signaling pathway
 ## G0:0022401 negative adaptation of signaling pathway
 ## G0:0023058 adaptation of signaling pathway
 ## G0:0097154 GABAergic neuron differentiation
 ## G0:0051923 sulfation
 ## G0:0021884 forebrain neuron development
 ## G0:0040037 negative regulation of fibroblast growth factor receptor signaling pathway
 ## G0:1901071 glucosamine-containing compound metabolic process
 ## G0:1904996 positive regulation of leukocyte adhesion to vascular endothelial cell
 ## G0:0021895 cerebral cortex neuron differentiation
 ## G0:1905564 positive regulation of vascular endothelial cell proliferation
 ## G0:0002092 positive regulation of receptor internalization
 ## G0:0050901 leukocyte tethering or rolling
 ## G0:0021799 cerebral cortex radially oriented cell migration
 ## G0:0045823 positive regulation of heart contraction
 ## G0:1903524 positive regulation of blood circulation
 ## G0:0016266 O-glycan processing
 ## G0:0040036 regulation of fibroblast growth factor receptor signaling pathway
 ## G0:1904994 regulation of leukocyte adhesion to vascular endothelial cell
 ## G0:0002691 regulation of cellular extravasation
 ## G0:1904706 negative regulation of vascular associated smooth muscle cell proliferation
 ## G0:0006040 amino sugar metabolic process
 ## G0:0007595 lactation
 ## G0:0021879 forebrain neuron differentiation
 ## G0:0021795 cerebral cortex cell migration
 ## G0:0060976 coronary vasculature development
 ## G0:0045776 negative regulation of blood pressure
 ## G0:1905562 regulation of vascular endothelial cell proliferation
 ## G0:0101023 vascular endothelial cell proliferation
 ## G0:0021872 forebrain generation of neurons
 ## G0:0048260 positive regulation of receptor-mediated endocytosis
 ## G0:1902895 positive regulation of miRNA transcription
 ## G0:0061756 leukocyte adhesion to vascular endothelial cell
 ## G0:0022029 telencephalon cell migration
 ## G0:0021885 forebrain cell migration
 ## G0:0045744 negative regulation of G protein-coupled receptor signaling pathway

G0:2000630 positive regulation of miRNA metabolic process
 ## G0:0048662 negative regulation of smooth muscle cell proliferation
 ## G0:0002090 regulation of receptor internalization
 ## G0:0045123 cellular extravasation
 ## G0:1902893 regulation of miRNA transcription
 ## G0:0061614 miRNA transcription
 ## G0:0008543 fibroblast growth factor receptor signaling pathway
 ## G0:2000628 regulation of miRNA metabolic process
 ## G0:0021954 central nervous system neuron development
 ## G0:1904705 regulation of vascular associated smooth muscle cell proliferation
 ## G0:0007589 body fluid secretion
 ## G0:1990874 vascular associated smooth muscle cell proliferation
 ## G0:0006493 protein O-linked glycosylation
 ## G0:0001938 positive regulation of endothelial cell proliferation
 ## G0:0007631 feeding behavior
 ## G0:0010586 miRNA metabolic process
 ## G0:0044344 cellular response to fibroblast growth factor stimulus
 ## G0:0048259 regulation of receptor-mediated endocytosis
 ## G0:0071774 response to fibroblast growth factor
 ## G0:0021987 cerebral cortex development
 ## G0:0030879 mammary gland development
 ## G0:0090288 negative regulation of cellular response to growth factor stimulus
 ## G0:0031623 receptor internalization
 ## G0:0045807 positive regulation of endocytosis
 ## G0:0008277 regulation of G protein-coupled receptor signaling pathway
 ## G0:0048660 regulation of smooth muscle cell proliferation
 ## G0:0048659 smooth muscle cell proliferation
 ## G0:0001936 regulation of endothelial cell proliferation
 ## G0:0001764 neuron migration
 ## G0:0008217 regulation of blood pressure
 ## G0:0021543 pallium development
 ## G0:0001935 endothelial cell proliferation
 ## G0:0007369 gastrulation
 ## G0:0021953 central nervous system neuron differentiation
 ## G0:0048469 cell maturation
 ## G0:0008016 regulation of heart contraction
 ## G0:0050679 positive regulation of epithelial cell proliferation
 ## G0:0006486 protein glycosylation
 ## G0:0043413 macromolecule glycosylation

## G0:0002685							regulation of leukocyte migration
## G0:0070085							glycosylation
## G0:0033002							muscle cell proliferation
## G0:0060047							heart contraction
## G0:0003015							heart process
## G0:1903522							regulation of blood circulation
## G0:0006898							receptor-mediated endocytosis
## G0:0071695							anatomical structure maturation
## G0:0021537							telencephalon development
## G0:1903039							positive regulation of leukocyte cell-cell adhesion
## G0:0030100							regulation of endocytosis
## G0:0009101							glycoprotein biosynthetic process
## G0:0022409							positive regulation of cell-cell adhesion
## G0:0006790							sulfur compound metabolic process
## G0:0021700							developmental maturation
## G0:0050878							regulation of body fluid levels
## G0:0090287							regulation of cellular response to growth factor stimulus
## G0:0009100							glycoprotein metabolic process
## G0:1903037							regulation of leukocyte cell-cell adhesion
## G0:0050900							leukocyte migration
## G0:0050678							regulation of epithelial cell proliferation
## G0:0030900							forebrain development
## G0:0007159							leukocyte cell-cell adhesion
## G0:0048732							gland development
## G0:0050673							epithelial cell proliferation
## G0:0045785							positive regulation of cell adhesion
## G0:0005802							trans-Golgi network
## G0:0098791							Golgi apparatus subcompartment
## G0:0071855							neuropeptide receptor binding
## G0:0008146							sulfotransferase activity
## G0:0016782							transferase activity, transferring sulphur-containing groups
## G0:0005179							hormone activity
## G0:0001664							G protein-coupled receptor binding
##	GeneRatio	BgRatio	RichFactor	FoldEnrichment	zScore	pvalue	
## G0:0021892	1/3	11/18986	0.090909091	575.33333	23.952617	0.001737207	
## G0:0042756	1/3	11/18986	0.090909091	575.33333	23.952617	0.001737207	
## G0:1903238	1/3	13/18986	0.076923077	486.82051	22.027389	0.002052847	
## G0:0002031	1/3	15/18986	0.066666667	421.91111	20.500970	0.002368420	
## G0:1904936	1/3	15/18986	0.066666667	421.91111	20.500970	0.002368420	

## G0:0006044	1/3	18/18986	0.055555556	351.59259	18.707326	0.002841655
## G0:1903236	1/3	18/18986	0.055555556	351.59259	18.707326	0.002841655
## G0:0002029	1/3	19/18986	0.052631579	333.08772	18.205969	0.002999367
## G0:0022401	1/3	20/18986	0.050000000	316.43333	17.742639	0.003157062
## G0:0023058	1/3	20/18986	0.050000000	316.43333	17.742639	0.003157062
## G0:0097154	1/3	20/18986	0.050000000	316.43333	17.742639	0.003157062
## G0:0051923	1/3	22/18986	0.045454545	287.66667	16.912471	0.003472402
## G0:0021884	1/3	25/18986	0.040000000	253.14667	15.859012	0.003945287
## G0:0040037	1/3	25/18986	0.040000000	253.14667	15.859012	0.003945287
## G0:1901071	1/3	25/18986	0.040000000	253.14667	15.859012	0.003945287
## G0:1904996	1/3	26/18986	0.038461538	243.41026	15.548983	0.004102883
## G0:0021895	1/3	27/18986	0.037037037	234.39506	15.256304	0.004260461
## G0:1905564	1/3	27/18986	0.037037037	234.39506	15.256304	0.004260461
## G0:0002092	1/3	29/18986	0.034482759	218.22989	14.716932	0.004575569
## G0:0050901	1/3	34/18986	0.029411765	186.13725	13.582799	0.005363047
## G0:0021799	1/3	37/18986	0.027027027	171.04505	13.015330	0.005835334
## G0:0045823	1/3	38/18986	0.026315789	166.54386	12.841232	0.005992730
## G0:1903524	1/3	39/18986	0.025641026	162.27350	12.673851	0.006150109
## G0:0016266	1/3	40/18986	0.025000000	158.21667	12.512766	0.006307472
## G0:0040036	1/3	41/18986	0.024390244	154.35772	12.357590	0.006464818
## G0:1904994	1/3	41/18986	0.024390244	154.35772	12.357590	0.006464818
## G0:0002691	1/3	42/18986	0.023809524	150.68254	12.207970	0.006622147
## G0:1904706	1/3	42/18986	0.023809524	150.68254	12.207970	0.006622147
## G0:0006040	1/3	43/18986	0.023255814	147.17829	12.063581	0.006779460
## G0:0007595	1/3	43/18986	0.023255814	147.17829	12.063581	0.006779460
## G0:0021879	1/3	45/18986	0.022222222	140.63704	11.789325	0.007094035
## G0:0021795	1/3	49/18986	0.020408163	129.15646	11.291886	0.007722988
## G0:0060976	1/3	50/18986	0.020000000	126.57333	11.176911	0.007880184
## G0:0045776	1/3	52/18986	0.019230769	121.70513	10.956951	0.008194528
## G0:1905562	1/3	52/18986	0.019230769	121.70513	10.956951	0.008194528
## G0:0101023	1/3	53/18986	0.018867925	119.40881	10.851649	0.008351674
## G0:0021872	1/3	55/18986	0.018181818	115.06667	10.649686	0.008665918
## G0:0048260	1/3	56/18986	0.017857143	113.01190	10.552768	0.008823015
## G0:1902895	1/3	56/18986	0.017857143	113.01190	10.552768	0.008823015
## G0:0061756	1/3	58/18986	0.017241379	109.11494	10.366469	0.009137159
## G0:0022029	1/3	62/18986	0.016129032	102.07527	10.021155	0.009765248
## G0:0021885	1/3	65/18986	0.015384615	97.36410	9.783256	0.010236141
## G0:0045744	1/3	66/18986	0.015151515	95.88889	9.707564	0.010393072
## G0:2000630	1/3	67/18986	0.014925373	94.45771	9.633563	0.010549986

## G0:0048662	1/3	70/18986	0.014285714	90.40952	9.421101	0.011020630
## G0:0002090	1/3	75/18986	0.013333333	84.38222	9.095579	0.011804704
## G0:0045123	1/3	75/18986	0.013333333	84.38222	9.095579	0.011804704
## G0:1902893	1/3	76/18986	0.013157895	83.27193	9.034336	0.011961469
## G0:0061614	1/3	77/18986	0.012987013	82.19048	8.974281	0.012118218
## G0:0008543	1/3	90/18986	0.011111111	70.31852	8.286461	0.014154441
## G0:2000628	1/3	92/18986	0.010869565	68.78986	8.193702	0.014467458
## G0:0021954	1/3	94/18986	0.010638298	67.32624	8.103896	0.014780408
## G0:1904705	1/3	94/18986	0.010638298	67.32624	8.103896	0.014780408
## G0:0007589	1/3	95/18986	0.010526316	66.61754	8.060051	0.014936858
## G0:1990874	1/3	96/18986	0.010416667	65.92361	8.016888	0.015093292
## G0:0006493	1/3	100/18986	0.010000000	63.28667	7.850704	0.015718861
## G0:0001938	1/3	107/18986	0.009345794	59.14642	7.582438	0.016812970
## G0:0007631	1/3	108/18986	0.009259259	58.59877	7.546239	0.016969205
## G0:0010586	1/3	110/18986	0.009090909	57.53333	7.475314	0.017281625
## G0:0044344	1/3	120/18986	0.008333333	52.73889	7.147453	0.018842734
## G0:0048259	1/3	128/18986	0.007812500	49.44271	6.913040	0.020090431
## G0:0071774	1/3	128/18986	0.007812500	49.44271	6.913040	0.020090431
## G0:0021987	1/3	130/18986	0.007692308	48.68205	6.857808	0.020402189
## G0:0030879	1/3	136/18986	0.007352941	46.53431	6.699403	0.021337069
## G0:0090288	1/3	139/18986	0.007194245	45.52998	6.624029	0.021804285
## G0:0031623	1/3	141/18986	0.007092199	44.88416	6.575106	0.022115680
## G0:0045807	1/3	159/18986	0.006289308	39.80294	6.176694	0.024915263
## G0:0008277	1/3	160/18986	0.006250000	39.55417	6.156527	0.025070638
## G0:0048660	1/3	174/18986	0.005747126	36.37165	5.892450	0.027244162
## G0:0048659	1/3	178/18986	0.005617978	35.55431	5.822700	0.027864575
## G0:0001936	1/3	179/18986	0.005586592	35.35568	5.805623	0.028019637
## G0:0001764	1/3	189/18986	0.005291005	33.48501	5.642260	0.029569350
## G0:0008217	1/3	190/18986	0.005263158	33.30877	5.626626	0.029724231
## G0:0021543	1/3	195/18986	0.005128205	32.45470	5.550236	0.030498387
## G0:0001935	1/3	202/18986	0.004950495	31.33003	5.448012	0.031581514
## G0:0007369	1/3	202/18986	0.004950495	31.33003	5.448012	0.031581514
## G0:0021953	1/3	203/18986	0.004926108	31.17570	5.433834	0.031736180
## G0:0048469	1/3	210/18986	0.004761905	30.13651	5.337392	0.032818385
## G0:0008016	1/3	213/18986	0.004694836	29.71205	5.297496	0.033281940
## G0:0050679	1/3	216/18986	0.004629630	29.29938	5.258419	0.033745347
## G0:0006486	1/3	223/18986	0.004484305	28.37967	5.170268	0.034826053
## G0:0043413	1/3	223/18986	0.004484305	28.37967	5.170268	0.034826053
## G0:0002685	1/3	236/18986	0.004237288	26.81638	5.016887	0.036830941

##	G0:0070085	1/3	242/18986	0.004132231	26.15152	4.950217	0.037755337
##	G0:0033002	1/3	252/18986	0.003968254	25.11376	4.844326	0.039294682
##	G0:0060047	1/3	253/18986	0.003952569	25.01449	4.834076	0.039448526
##	G0:0003015	1/3	262/18986	0.003816794	24.15522	4.744426	0.040832384
##	G0:1903522	1/3	267/18986	0.003745318	23.70287	4.696546	0.041600619
##	G0:0006898	1/3	275/18986	0.003636364	23.01333	4.622609	0.042828943
##	G0:0071695	1/3	276/18986	0.003623188	22.92995	4.613588	0.042982409
##	G0:0021537	1/3	281/18986	0.003558719	22.52195	4.569191	0.043749496
##	G0:1903039	1/3	284/18986	0.003521127	22.28404	4.543104	0.044209551
##	G0:0030100	1/3	307/18986	0.003257329	20.61455	4.355662	0.047731741
##	G0:0009101	1/3	322/18986	0.003105590	19.65424	4.244111	0.050024153
##	G0:0022409	1/3	336/18986	0.002976190	18.83532	4.146623	0.052160415
##	G0:0006790	1/3	338/18986	0.002958580	18.72387	4.133179	0.052465334
##	G0:0021700	1/3	350/18986	0.002857143	18.08190	4.054875	0.054293472
##	G0:0050878	1/3	381/18986	0.002624672	16.61067	3.869482	0.059005277
##	G0:0090287	1/3	381/18986	0.002624672	16.61067	3.869482	0.059005277
##	G0:0009100	1/3	392/18986	0.002551020	16.14456	3.808875	0.060673438
##	G0:1903037	1/3	392/18986	0.002551020	16.14456	3.808875	0.060673438
##	G0:0050900	1/3	404/18986	0.002475248	15.66502	3.745506	0.062491000
##	G0:0050678	1/3	406/18986	0.002463054	15.58785	3.735209	0.062793699
##	G0:0030900	1/3	416/18986	0.002403846	15.21314	3.684803	0.064306216
##	G0:0007159	1/3	430/18986	0.002325581	14.71783	3.617104	0.066421005
##	G0:0048732	1/3	450/18986	0.002222222	14.06370	3.525722	0.069436601
##	G0:0050673	1/3	482/18986	0.002074689	13.13001	3.391060	0.074248036
##	G0:0045785	1/3	499/18986	0.002004008	12.68270	3.324633	0.076797352
##	G0:0005802	1/3	269/19960	0.003717472	24.73358	4.804939	0.039890396
##	G0:0098791	1/3	391/19960	0.002557545	17.01620	3.921443	0.057626675
##	G0:0071855	1/3	38/18737	0.026315789	164.35965	12.755903	0.006072211
##	G0:0008146	1/3	53/18737	0.018867925	117.84277	10.779253	0.008462353
##	G0:0016782	1/3	71/18737	0.014084507	87.96714	9.290573	0.011325462
##	G0:0005179	1/3	132/18737	0.007575758	47.31566	6.757464	0.020987224
##	G0:0001664	1/3	299/18737	0.003344482	20.88852	4.386972	0.047115783
##		p.adjust	qvalue	geneID	Count		
##	G0:0021892	0.02440605	0.003568137	26468	1		
##	G0:0042756	0.02440605	0.003568137	8862	1		
##	G0:1903238	0.02440605	0.003568137	10164	1		
##	G0:0002031	0.02440605	0.003568137	8862	1		
##	G0:1904936	0.02440605	0.003568137	26468	1		
##	G0:0006044	0.02440605	0.003568137	10164	1		

##	G0:1903236	0.02440605	0.003568137	10164	1
##	G0:0002029	0.02440605	0.003568137	8862	1
##	G0:0022401	0.02440605	0.003568137	8862	1
##	G0:0023058	0.02440605	0.003568137	8862	1
##	G0:0097154	0.02440605	0.003568137	26468	1
##	G0:0051923	0.02440605	0.003568137	10164	1
##	G0:0021884	0.02440605	0.003568137	26468	1
##	G0:0040037	0.02440605	0.003568137	8862	1
##	G0:1901071	0.02440605	0.003568137	10164	1
##	G0:1904996	0.02440605	0.003568137	10164	1
##	G0:0021895	0.02440605	0.003568137	26468	1
##	G0:1905564	0.02440605	0.003568137	8862	1
##	G0:0002092	0.02440605	0.003568137	8862	1
##	G0:0050901	0.02440605	0.003568137	10164	1
##	G0:0021799	0.02440605	0.003568137	26468	1
##	G0:0045823	0.02440605	0.003568137	8862	1
##	G0:1903524	0.02440605	0.003568137	8862	1
##	G0:0016266	0.02440605	0.003568137	10164	1
##	G0:0040036	0.02440605	0.003568137	8862	1
##	G0:1904994	0.02440605	0.003568137	10164	1
##	G0:0002691	0.02440605	0.003568137	10164	1
##	G0:1904706	0.02440605	0.003568137	8862	1
##	G0:0006040	0.02440605	0.003568137	10164	1
##	G0:0007595	0.02440605	0.003568137	8862	1
##	G0:0021879	0.02443296	0.003572071	26468	1
##	G0:0021795	0.02443296	0.003572071	26468	1
##	G0:0060976	0.02443296	0.003572071	8862	1
##	G0:0045776	0.02443296	0.003572071	8862	1
##	G0:1905562	0.02443296	0.003572071	8862	1
##	G0:0101023	0.02443296	0.003572071	8862	1
##	G0:0021872	0.02443296	0.003572071	26468	1
##	G0:0048260	0.02443296	0.003572071	8862	1
##	G0:1902895	0.02443296	0.003572071	8862	1
##	G0:0061756	0.02467033	0.003606773	10164	1
##	G0:0022029	0.02572309	0.003760686	26468	1
##	G0:0021885	0.02589542	0.003785880	26468	1
##	G0:0045744	0.02589542	0.003785880	8862	1
##	G0:2000630	0.02589542	0.003785880	8862	1
##	G0:0048662	0.02644951	0.003866888	8862	1

##	G0:0002090	0.02670954	0.003904904	8862	1
##	G0:0045123	0.02670954	0.003904904	10164	1
##	G0:1902893	0.02670954	0.003904904	8862	1
##	G0:0061614	0.02670954	0.003904904	8862	1
##	G0:0008543	0.02963774	0.004333002	8862	1
##	G0:2000628	0.02963774	0.004333002	8862	1
##	G0:0021954	0.02963774	0.004333002	26468	1
##	G0:1904705	0.02963774	0.004333002	8862	1
##	G0:0007589	0.02963774	0.004333002	8862	1
##	G0:1990874	0.02963774	0.004333002	8862	1
##	G0:0006493	0.03031495	0.004432010	10164	1
##	G0:0001938	0.03159783	0.004619566	8862	1
##	G0:0007631	0.03159783	0.004619566	8862	1
##	G0:0010586	0.03163416	0.004624877	8862	1
##	G0:0044344	0.03391692	0.004958614	8862	1
##	G0:0048259	0.03497518	0.005113331	8862	1
##	G0:0071774	0.03497518	0.005113331	8862	1
##	G0:0021987	0.03497518	0.005113331	26468	1
##	G0:0030879	0.03600630	0.005264079	8862	1
##	G0:0090288	0.03618930	0.005290833	8862	1
##	G0:0031623	0.03618930	0.005290833	8862	1
##	G0:0045807	0.03981807	0.005821356	8862	1
##	G0:0008277	0.03981807	0.005821356	8862	1
##	G0:0048660	0.04262142	0.006231202	8862	1
##	G0:0048659	0.04262142	0.006231202	8862	1
##	G0:0001936	0.04262142	0.006231202	8862	1
##	G0:0001764	0.04397557	0.006429178	26468	1
##	G0:0008217	0.04397557	0.006429178	8862	1
##	G0:0021543	0.04451116	0.006507480	26468	1
##	G0:0001935	0.04451308	0.006507761	8862	1
##	G0:0007369	0.04451308	0.006507761	8862	1
##	G0:0021953	0.04451308	0.006507761	26468	1
##	G0:0048469	0.04544084	0.006643398	26468	1
##	G0:0008016	0.04549936	0.006651953	8862	1
##	G0:0050679	0.04555622	0.006660266	8862	1
##	G0:0006486	0.04586846	0.006705915	10164	1
##	G0:0043413	0.04586846	0.006705915	10164	1
##	G0:0002685	0.04792460	0.007006520	10164	1
##	G0:0070085	0.04854258	0.007096868	10164	1

##	G0:0033002	0.04954001	0.007242691	8862	1
##	G0:0060047	0.04954001	0.007242691	8862	1
##	G0:0003015	0.05068848	0.007410596	8862	1
##	G0:1903522	0.05105531	0.007464226	8862	1
##	G0:0006898	0.05157889	0.007540774	8862	1
##	G0:0071695	0.05157889	0.007540774	26468	1
##	G0:0021537	0.05189817	0.007587452	26468	1
##	G0:1903039	0.05189817	0.007587452	10164	1
##	G0:0030100	0.05543041	0.008103861	8862	1
##	G0:0009101	0.05747456	0.008402713	10164	1
##	G0:0022409	0.05902350	0.008629167	10164	1
##	G0:0006790	0.05902350	0.008629167	10164	1
##	G0:0021700	0.06045046	0.008837787	26468	1
##	G0:0050878	0.06436939	0.009410730	8862	1
##	G0:0090287	0.06436939	0.009410730	8862	1
##	G0:0009100	0.06487853	0.009485165	10164	1
##	G0:1903037	0.06487853	0.009485165	10164	1
##	G0:0050900	0.06584194	0.009626014	10164	1
##	G0:0050678	0.06584194	0.009626014	8862	1
##	G0:0030900	0.06677953	0.009763090	26468	1
##	G0:0007159	0.06831875	0.009988121	10164	1
##	G0:0048732	0.07074673	0.010343089	8862	1
##	G0:0050673	0.07494194	0.010956424	8862	1
##	G0:0045785	0.07679735	0.011227683	10164	1
##	G0:0005802	0.05762667	0.030329829	10164	1
##	G0:0098791	0.05762667	0.030329829	10164	1
##	G0:0071855	0.01887577	NA	8862	1
##	G0:0008146	0.01887577	NA	10164	1
##	G0:0016782	0.01887577	NA	10164	1
##	G0:0005179	0.02623403	NA	8862	1
##	G0:0001664	0.04711578	NA	8862	1

Los resultados de enriquecimiento en Gene Ontology (GO) para los genes seleccionados indican que estos genes están asociados principalmente con **procesos biológicos** (BP), **componentes celulares** (CC) y **funciones moleculares** (MF). En términos de procesos biológicos, algunos de los términos más enriquecidos incluyen la *diferenciación de interneuronas GABAérgicas en la corteza cerebral*, el *comportamiento de consumo (de líquidos)*, la *vía de señalización del receptor del factor de crecimiento de hepatocitos* y la *regulación positiva del anclaje o rodamiento de leucocitos*. Los genes seleccionados tienen un papel importante en una variedad de procesos biológicos, lo que podría indicar que están involucrados

en la regulación de funciones clave relacionadas con el sistema nervioso, el sistema inmune y el sistema hepático. Estas asociaciones pueden proporcionar pistas importantes para futuras investigaciones sobre el desarrollo de terapias para diversas condiciones médicas.

En cuanto a los **componentes celulares**, se observan asociaciones con la *red del trans-Golgi* y el *subcompartimiento del aparato de Golgi*, lo que sugiere que los genes seleccionados intervienen en el procesamiento y tráfico de proteínas dentro de la célula. La red del trans-Golgi juega un papel importante en la distribución de proteínas y lípidos hacia sus destinos dentro y/o fuera de la célula, mientras que el subcompartimiento del aparato de Golgi se encarga de la modificación y empaquetado de estas moléculas. Estos “descubrimientos” podrán indicar que los genes seleccionados podrían estar involucrados en la regulación de procesos de transporte vesicular, modificación de proteínas y mantenimiento de la organización intracelular, lo cual es algo importante para el funcionamiento adecuado de la célula.

Por último, en cuanto a las **funciones moleculares**, se caracterizan la *unión a receptores de neuropeptidos*, la *actividad sulfotransferasa*, la *actividad transferasa*, transfiriendo grupos que contienen azufre, la *actividad hormonal* y la *unión a receptores de factores de crecimiento*, entre otros. Estos “descubrimientos” sugieren que los genes seleccionados podrían estar involucrados en la modulación de señales neurobiológicas, la regulación de la transferencia de grupos químicos esenciales como los sulfuros, la regulación hormonal y el control del crecimiento celular, lo que podría implicar roles importantes en la comunicación celular y el mantenimiento de la homeostasis en el organismo.

Una vez descritos los resultados del Gene Ontology pasaremos a ver cuales son las ontologías más enriquecidas, para determinarlos, habría que observar el GeneRatio, que indica la proporción de genes en la lista asociados con cada término en relación al total de genes de la lista. Sin embargo, en este caso, todos los términos tienen el mismo valor de GeneRatio (1/3), por lo que este criterio no permite diferenciar entre ellos, ya que todos están enriquecidos.

En consecuencia, el RichFactor se utiliza como el siguiente criterio para identificar los términos más enriquecidos. El RichFactor mide la proporción entre el número de genes en una categoría específica (GeneRatio) y el número total de genes de fondo en esa categoría (BgRatio), por lo que estarían ordenados de mayor a menor. Los términos con el RichFactor más alto son:

- **GO:0021892** = 0.090909091
- **GO:0042756** = 0.090909091
- **GO:1903238** = 0.076923077
- **GO:0002031** = 0.066666667
- **GO:1904936** = 0.066666667

Estos términos presentan el mayor RichFactor y, por lo tanto, pueden considerarse los más enriquecidos. Sin embargo, es importante considerar otros indicadores como el p-value ajustado para confirmar la significancia estadística del enriquecimiento, especialmente si los términos tienen valores similares.

Se ha decidido utilizar el *p.adjust* ya que corrige el problema de las comparaciones múltiples, reduciendo el número de falsos positivos.

En vez de utilizar el umbral común 0.05, se ha decidido utilizar uno menor ya que con ese umbral aparece que son significativos la mayoría de las muestras por lo que no podríamos determinar los genes más enriquecidos por lo que se ha decidido utilizar **0.025** como umbral.

```
G0s_significativos <- G0s[G0s$p.adjust < 0.025, ]
```

```
G0s_significativos
```

##	ONTOLOGY	ID
## G0:0021892	BP	G0:0021892
## G0:0042756	BP	G0:0042756
## G0:1903238	BP	G0:1903238
## G0:0002031	BP	G0:0002031
## G0:1904936	BP	G0:1904936
## G0:0006044	BP	G0:0006044
## G0:1903236	BP	G0:1903236
## G0:0002029	BP	G0:0002029
## G0:0022401	BP	G0:0022401
## G0:0023058	BP	G0:0023058
## G0:0097154	BP	G0:0097154
## G0:0051923	BP	G0:0051923
## G0:0021884	BP	G0:0021884
## G0:0040037	BP	G0:0040037
## G0:1901071	BP	G0:1901071
## G0:1904996	BP	G0:1904996
## G0:0021895	BP	G0:0021895
## G0:1905564	BP	G0:1905564
## G0:0002092	BP	G0:0002092
## G0:0050901	BP	G0:0050901
## G0:0021799	BP	G0:0021799
## G0:0045823	BP	G0:0045823
## G0:1903524	BP	G0:1903524
## G0:0016266	BP	G0:0016266
## G0:0040036	BP	G0:0040036
## G0:1904994	BP	G0:1904994
## G0:0002691	BP	G0:0002691
## G0:1904706	BP	G0:1904706

## G0:0006040	BP G0:0006040
## G0:0007595	BP G0:0007595
## G0:0021879	BP G0:0021879
## G0:0021795	BP G0:0021795
## G0:0060976	BP G0:0060976
## G0:0045776	BP G0:0045776
## G0:1905562	BP G0:1905562
## G0:0101023	BP G0:0101023
## G0:0021872	BP G0:0021872
## G0:0048260	BP G0:0048260
## G0:1902895	BP G0:1902895
## G0:0061756	BP G0:0061756
## G0:0071855	MF G0:0071855
## G0:0008146	MF G0:0008146
## G0:0016782	MF G0:0016782

##	Description
## G0:0021892	cerebral cortex GABAergic interneuron differentiation
## G0:0042756	drinking behavior
## G0:1903238	positive regulation of leukocyte tethering or rolling
## G0:0002031	G protein-coupled receptor internalization
## G0:1904936	interneuron migration
## G0:0006044	N-acetylglucosamine metabolic process
## G0:1903236	regulation of leukocyte tethering or rolling
## G0:0002029	desensitization of G protein-coupled receptor signaling pathway
## G0:0022401	negative adaptation of signaling pathway
## G0:0023058	adaptation of signaling pathway
## G0:0097154	GABAergic neuron differentiation
## G0:0051923	sulfation
## G0:0021884	forebrain neuron development
## G0:0040037	negative regulation of fibroblast growth factor receptor signaling pathway
## G0:1901071	glucosamine-containing compound metabolic process
## G0:1904996	positive regulation of leukocyte adhesion to vascular endothelial cell
## G0:0021895	cerebral cortex neuron differentiation
## G0:1905564	positive regulation of vascular endothelial cell proliferation
## G0:0002092	positive regulation of receptor internalization
## G0:0050901	leukocyte tethering or rolling
## G0:0021799	cerebral cortex radially oriented cell migration
## G0:0045823	positive regulation of heart contraction
## G0:1903524	positive regulation of blood circulation

## G0:0016266		O-glycan processing
## G0:0040036	regulation of fibroblast growth factor receptor signaling pathway	
## G0:1904994	regulation of leukocyte adhesion to vascular endothelial cell	
## G0:0002691	regulation of cellular extravasation	
## G0:1904706	negative regulation of vascular associated smooth muscle cell proliferation	
## G0:0006040	amino sugar metabolic process	
## G0:0007595	lactation	
## G0:0021879	forebrain neuron differentiation	
## G0:0021795	cerebral cortex cell migration	
## G0:0060976	coronary vasculature development	
## G0:0045776	negative regulation of blood pressure	
## G0:1905562	regulation of vascular endothelial cell proliferation	
## G0:0101023	vascular endothelial cell proliferation	
## G0:0021872	forebrain generation of neurons	
## G0:0048260	positive regulation of receptor-mediated endocytosis	
## G0:1902895	positive regulation of miRNA transcription	
## G0:0061756	leukocyte adhesion to vascular endothelial cell	
## G0:0071855	neuropeptide receptor binding	
## G0:0008146	sulfotransferase activity	
## G0:0016782	transferase activity, transferring sulphur-containing groups	
##	GeneRatio BgRatio RichFactor FoldEnrichment zScore pvalue	
## G0:0021892	1/3 11/18986 0.09090909 575.33333 23.952617 0.001737207	
## G0:0042756	1/3 11/18986 0.09090909 575.33333 23.952617 0.001737207	
## G0:1903238	1/3 13/18986 0.07692308 486.82051 22.027389 0.002052847	
## G0:0002031	1/3 15/18986 0.06666667 421.91111 20.500970 0.002368420	
## G0:1904936	1/3 15/18986 0.06666667 421.91111 20.500970 0.002368420	
## G0:0006044	1/3 18/18986 0.05555556 351.59259 18.707326 0.002841655	
## G0:1903236	1/3 18/18986 0.05555556 351.59259 18.707326 0.002841655	
## G0:0002029	1/3 19/18986 0.05263158 333.08772 18.205969 0.002999367	
## G0:0022401	1/3 20/18986 0.05000000 316.43333 17.742639 0.003157062	
## G0:0023058	1/3 20/18986 0.05000000 316.43333 17.742639 0.003157062	
## G0:0097154	1/3 20/18986 0.05000000 316.43333 17.742639 0.003157062	
## G0:0051923	1/3 22/18986 0.04545455 287.66667 16.912471 0.003472402	
## G0:0021884	1/3 25/18986 0.04000000 253.14667 15.859012 0.003945287	
## G0:0040037	1/3 25/18986 0.04000000 253.14667 15.859012 0.003945287	
## G0:1901071	1/3 25/18986 0.04000000 253.14667 15.859012 0.003945287	
## G0:1904996	1/3 26/18986 0.03846154 243.41026 15.548983 0.004102883	
## G0:0021895	1/3 27/18986 0.03703704 234.39506 15.256304 0.004260461	
## G0:1905564	1/3 27/18986 0.03703704 234.39506 15.256304 0.004260461	

##	G0:0002092	1/3	29/18986	0.03448276	218.22989	14.716932	0.004575569
##	G0:0050901	1/3	34/18986	0.02941176	186.13725	13.582799	0.005363047
##	G0:0021799	1/3	37/18986	0.02702703	171.04505	13.015330	0.005835334
##	G0:0045823	1/3	38/18986	0.02631579	166.54386	12.841232	0.005992730
##	G0:1903524	1/3	39/18986	0.02564103	162.27350	12.673851	0.006150109
##	G0:0016266	1/3	40/18986	0.02500000	158.21667	12.512766	0.006307472
##	G0:0040036	1/3	41/18986	0.02439024	154.35772	12.357590	0.006464818
##	G0:1904994	1/3	41/18986	0.02439024	154.35772	12.357590	0.006464818
##	G0:0002691	1/3	42/18986	0.02380952	150.68254	12.207970	0.006622147
##	G0:1904706	1/3	42/18986	0.02380952	150.68254	12.207970	0.006622147
##	G0:0006040	1/3	43/18986	0.02325581	147.17829	12.063581	0.006779460
##	G0:0007595	1/3	43/18986	0.02325581	147.17829	12.063581	0.006779460
##	G0:0021879	1/3	45/18986	0.02222222	140.63704	11.789325	0.007094035
##	G0:0021795	1/3	49/18986	0.02040816	129.15646	11.291886	0.007722988
##	G0:0060976	1/3	50/18986	0.02000000	126.57333	11.176911	0.007880184
##	G0:0045776	1/3	52/18986	0.01923077	121.70513	10.956951	0.008194528
##	G0:1905562	1/3	52/18986	0.01923077	121.70513	10.956951	0.008194528
##	G0:0101023	1/3	53/18986	0.01886792	119.40881	10.851649	0.008351674
##	G0:0021872	1/3	55/18986	0.01818182	115.06667	10.649686	0.008665918
##	G0:0048260	1/3	56/18986	0.01785714	113.01190	10.552768	0.008823015
##	G0:1902895	1/3	56/18986	0.01785714	113.01190	10.552768	0.008823015
##	G0:0061756	1/3	58/18986	0.01724138	109.11494	10.366469	0.009137159
##	G0:0071855	1/3	38/18737	0.02631579	164.35965	12.755903	0.006072211
##	G0:0008146	1/3	53/18737	0.01886792	117.84277	10.779253	0.008462353
##	G0:0016782	1/3	71/18737	0.01408451	87.96714	9.290573	0.011325462
##		p.adjust	qvalue	geneID	Count		
##	G0:0021892	0.02440605	0.003568137	26468	1		
##	G0:0042756	0.02440605	0.003568137	8862	1		
##	G0:1903238	0.02440605	0.003568137	10164	1		
##	G0:0002031	0.02440605	0.003568137	8862	1		
##	G0:1904936	0.02440605	0.003568137	26468	1		
##	G0:0006044	0.02440605	0.003568137	10164	1		
##	G0:1903236	0.02440605	0.003568137	10164	1		
##	G0:0002029	0.02440605	0.003568137	8862	1		
##	G0:0022401	0.02440605	0.003568137	8862	1		
##	G0:0023058	0.02440605	0.003568137	8862	1		
##	G0:0097154	0.02440605	0.003568137	26468	1		
##	G0:0051923	0.02440605	0.003568137	10164	1		
##	G0:0021884	0.02440605	0.003568137	26468	1		

##	G0:0040037	0.02440605	0.003568137	8862	1
##	G0:1901071	0.02440605	0.003568137	10164	1
##	G0:1904996	0.02440605	0.003568137	10164	1
##	G0:0021895	0.02440605	0.003568137	26468	1
##	G0:1905564	0.02440605	0.003568137	8862	1
##	G0:0002092	0.02440605	0.003568137	8862	1
##	G0:0050901	0.02440605	0.003568137	10164	1
##	G0:0021799	0.02440605	0.003568137	26468	1
##	G0:0045823	0.02440605	0.003568137	8862	1
##	G0:1903524	0.02440605	0.003568137	8862	1
##	G0:0016266	0.02440605	0.003568137	10164	1
##	G0:0040036	0.02440605	0.003568137	8862	1
##	G0:1904994	0.02440605	0.003568137	10164	1
##	G0:0002691	0.02440605	0.003568137	10164	1
##	G0:1904706	0.02440605	0.003568137	8862	1
##	G0:0006040	0.02440605	0.003568137	10164	1
##	G0:0007595	0.02440605	0.003568137	8862	1
##	G0:0021879	0.02443296	0.003572071	26468	1
##	G0:0021795	0.02443296	0.003572071	26468	1
##	G0:0060976	0.02443296	0.003572071	8862	1
##	G0:0045776	0.02443296	0.003572071	8862	1
##	G0:1905562	0.02443296	0.003572071	8862	1
##	G0:0101023	0.02443296	0.003572071	8862	1
##	G0:0021872	0.02443296	0.003572071	26468	1
##	G0:0048260	0.02443296	0.003572071	8862	1
##	G0:1902895	0.02443296	0.003572071	8862	1
##	G0:0061756	0.02467033	0.003606773	10164	1
##	G0:0071855	0.01887577	NA	8862	1
##	G0:0008146	0.01887577	NA	10164	1
##	G0:0016782	0.01887577	NA	10164	1

Una vez que hemos obtenido los más significativos dado el umbral que hemos definido anteriormente, los vamos a ordenar, en orden desdeciente (los resultados del RichFactor ya están ordenados pero se ordenarán igualmente para confirmar) por FoldEnrichment y RichFactor, con el objetivo de que podamos confirmar que hemos obtenido los mismos genes que comentamos anteriormente mirando el RichFactor solo, sin tener en cuenta el p.adjusted como en este proceso.

```
# Ordenar por FoldEnrichment en orden descendente
G0s_significativosFoldEnrichment<-G0s_significativos[
  order(-G0s_significativos$FoldEnrichment),]
```

```
# Ordenar por RichFactor en orden descendente
```

```
G0s_significativosRichFactor<-G0s_significativos[order(-G0s_significativos$RichFactor),]
```

Tras ordenar los resultados, seleccionaremos los 4 ontologías más significativas. Este número ha sido elegido porque corresponde a los 4 genes utilizados para la huella. Sin embargo, también podríamos analizar los 10 genes más significativos u optar por otro número, dependiendo del enfoque del análisis.

```
# Quedarnos con 4
```

```
G0s_top4_FE <- head(G0s_significativosFoldEnrichment, 4)
```

```
G0s_top4_FE
```

```
##          ONTOLOGY          ID
## G0:0021892      BP G0:0021892
## G0:0042756      BP G0:0042756
## G0:1903238      BP G0:1903238
## G0:0002031      BP G0:0002031
##
##                                     Description GeneRatio
## G0:0021892 cerebral cortex GABAergic interneuron differentiation      1/3
## G0:0042756                                     drinking behavior      1/3
## G0:1903238 positive regulation of leukocyte tethering or rolling      1/3
## G0:0002031          G protein-coupled receptor internalization      1/3
##          BgRatio RichFactor FoldEnrichment   zScore    pvalue   p.adjust
## G0:0021892 11/18986 0.09090909      575.3333 23.95262 0.001737207 0.02440605
## G0:0042756 11/18986 0.09090909      575.3333 23.95262 0.001737207 0.02440605
## G0:1903238 13/18986 0.07692308      486.8205 22.02739 0.002052847 0.02440605
## G0:0002031 15/18986 0.06666667      421.9111 20.50097 0.002368420 0.02440605
##
##          qvalue geneID Count
## G0:0021892 0.003568137 26468    1
## G0:0042756 0.003568137  8862    1
## G0:1903238 0.003568137 10164    1
## G0:0002031 0.003568137  8862    1
```

```
# Quedarnos con 4
```

```
G0s_top4_RF <- head(G0s_significativosRichFactor, 4)
```

```
G0s_top4_RF
```

```
##          ONTOLOGY          ID
## G0:0021892      BP G0:0021892
## G0:0042756      BP G0:0042756
## G0:1903238      BP G0:1903238
```

```
## GO:0002031      BP GO:0002031
##
##                                     Description GeneRatio
## GO:0021892 cerebral cortex GABAergic interneuron differentiation      1/3
## GO:0042756                                     drinking behavior      1/3
## GO:1903238 positive regulation of leukocyte tethering or rolling      1/3
## GO:0002031      G protein-coupled receptor internalization      1/3
##          BgRatio RichFactor FoldEnrichment   zScore      pvalue   p.adjust
## GO:0021892 11/18986 0.09090909      575.3333 23.95262 0.001737207 0.02440605
## GO:0042756 11/18986 0.09090909      575.3333 23.95262 0.001737207 0.02440605
## GO:1903238 13/18986 0.07692308      486.8205 22.02739 0.002052847 0.02440605
## GO:0002031 15/18986 0.06666667      421.9111 20.50097 0.002368420 0.02440605
##          qvalue geneID Count
## GO:0021892 0.003568137 26468      1
## GO:0042756 0.003568137 8862      1
## GO:1903238 0.003568137 10164      1
## GO:0002031 0.003568137 8862      1
```

Se comprueba que el criterio seguido anteriormente con RichFactor nos propociona el mismo resultado que el que acabamos de hacer, filtrando el *p.adjusted* con un umbral y ordenandolos por el valor de Fold-Enrichment mayor. El Fold Enrichment es una métrica común en análisis de enriquecimiento funcional (como GO o KEGG) que mide como de “enriquecida” está una categoría (como un término GO o una vía) en comparación con lo esperado por azar.

Por tanto, tras estos dos estudios podemos determinar que los términos GO más significativos asociados a los genes son:

- **GO:0021892**
- **GO:0042756**
- **GO:1903238**
- **GO:0002031**

7.2 Pathways

A continuación, obtenemos información de Pathways:

```
pathways <- DEGsToPathways(entrezAnnotation$external_gene_name)
```

Muestro información sobre los Pathways:

```
pathways_df <- data.frame(
  Kegg_path = unlist(pathways$KEGG_Path),
  Name = unlist(pathways$Name),
  Description = unlist(pathways$Description),
  Class = unlist(pathways$Class),
  Genes = unlist(pathways$Genes)
)
```

```
pathways_df
```

```
##   Kegg_path           Name
## 1  map04080           nothing
## 2  map04371 Apelin signaling pathway
##
## 1
## 2 Apelin is an endogenous peptide capable of binding the apelin receptor (APJ), which was original
##
##                               Class
## 1 Environmental Information Processing; Signaling molecules and interaction
## 2                               Environmental Information Processing; Signal transduction
##   Genes
## 1  APLN
## 2  APLN
```

```
pathways_df$Description
```

```
## [1] "nothing"
## [2] "Apelin is an endogenous peptide capable of binding the apelin receptor (APJ), which was original"
```

Los resultados obtenidos muestran que los genes seleccionados están asociados con diversas rutas metabólicas y de señalización, indicando que están implicados en de procesos biológicos. Estas rutas son importantes para mantener el equilibrio del organismo y regular funciones críticas, como el metabolismo y la comunicación celular.

- **APLN (Apelin):** El gen APLN está vinculado a la ruta map04371 relacionada con la señalización del receptor del péptido Apelin, que tiene un papel en procesos fisiológicos como la *angiogénesis* (formación de nuevos vasos sanguíneos), las *funciones cardiovasculares* (como la regulación de la presión arterial y la contractilidad del corazón), la *proliferación celular* (necesaria para la regeneración de tejidos) y la *regulación del metabolismo energético* (afectando el balance de glucosa y lípidos en el cuerpo). Además, esta vía está asociada con enfermedades como la diabetes, la obesidad, las enfermedades cardiovasculares y el cáncer, lo que refleja la importancia de APLN en la

regulación de la función cardiovascular y el control metabólico. También destaca su potencial como un objetivo terapéutico en el tratamiento de trastornos relacionados con el sistema cardiovascular y el metabolismo.

7.3 Disease Association

Por último, veremos las enfermedades a las que están asociados los genes:

```
diseases <- DEGsToDiseases(entrezAnnotation$external_gene_name,getEvidences=TRUE)
```

7.3.1 LINP1

```
diseases$LINP1
```

```
## $summary
##      Disease                                Overall Score
## [1,] "triple-negative breast cancer"         "0.0848819597376444"
## [2,] "glioma"                                "0.0765262313217218"
## [3,] "esophageal squamous cell carcinoma"    "0.0755585480688724"
## [4,] "neoplasm"                              "0.0655837363646982"
## [5,] "lung cancer"                           "0.057980508891502"
## [6,] "breast cancer"                         "0.0530717906784123"
## [7,] "cervical cancer"                       "0.0516422134861614"
## [8,] "skin squamous cell carcinoma"          "0.0510020199464888"
## [9,] "osteosarcoma"                          "0.0502423280229703"
## [10,] "hepatocellular carcinoma"             "0.0485637439917621"
##      Literature      RNA Expr.      Genetic Assoc. Somatic Mut.
## [1,] "0.698121892089695" "0"      "0"      "0"
## [2,] "0.629399198250612" "0"      "0"      "0"
## [3,] "0.621440370891879" "0"      "0"      "0"
## [4,] "0.539401331717007" "0"      "0"      "0"
## [5,] "0.476867672433196" "0"      "0"      "0"
## [6,] "0.436495328801531" "0"      "0"      "0"
## [7,] "0.424737599156419" "0"      "0"      "0"
## [8,] "0.419472250352019" "0"      "0"      "0"
## [9,] "0.413224072709899" "0"      "0"      "0"
## [10,] "0.380298403195418" "0.076479812633018" "0"      "0"
##      Known Drug Animal Model Affected Pathways
```

##	[1,]	"0"	"0"	"0"
##	[2,]	"0"	"0"	"0"
##	[3,]	"0"	"0"	"0"
##	[4,]	"0"	"0"	"0"
##	[5,]	"0"	"0"	"0"
##	[6,]	"0"	"0"	"0"
##	[7,]	"0"	"0"	"0"
##	[8,]	"0"	"0"	"0"
##	[9,]	"0"	"0"	"0"
##	[10,]	"0"	"0"	"0"

El gen **LINP1** (LncRNA In Non-Homologous End Joining Pathway 1) está asociado con varias enfermedades, aunque con una puntuación de asociación bastante bajo en algunos casos. Entre las enfermedades relacionadas con LINP1, destacan:

- **Cáncer de esófago (Carcinoma escamoso)**, con una puntuación de 0.074, lo que sugiere una asociación moderada entre el gen y este tipo de cáncer.
- **Neoplasia**, con una puntuación de 0.063, indica una posible relación con diversas formas de tumor.
- **Carcinoma de células escamosas de la piel**, con una puntuación de 0.051, lo que refleja una asociación más débil pero relevante.
- **Cáncer de mama triple negativo y otros cánceres como leucemia mieloide aguda y carcinoma papilar de tiroides** también están en la lista, pero con puntuaciones más bajas (0.035 y 0.031).

Es importante destacar que las principales evidencias sobre la relación de LINP1 con estas enfermedades provienen de estudios sobre la literatura científica y la expresión de ARN. Por otro lado, no se han encontrado asociaciones claras con mutaciones genéticas, medicamentos o modelos animales en este caso.

En una página de GeneCards (<https://www.genecards.org/cgi-bin/carddisp.pl?gene=LINP1&keywords=LINP1>), se menciona que LINP1 es un gen de ARN largo no codificante (lncRNA) relacionado con el cáncer cervical y el cáncer de mama. Esta información apoya nuestras asociaciones y ayuda a entender mejor cómo LINP1 podría estar involucrado en el desarrollo del cáncer.

7.3.2 LHX6

diseases\$**LHX6**

\$summary

##	Disease	Overall Score	Literature
----	---------	---------------	------------

```

## [1,] "body height"          "0.332907427009865" "0"
## [2,] "Alzheimer disease"    "0.276739753347064" "0.0303965398805811"
## [3,] "Parkinson disease"    "0.275815803710352" "0"
## [4,] "neurodegenerative disease" "0.275815803710352" "0"
## [5,] "multiple sclerosis"    "0.275815803710352" "0"
## [6,] "lysosomal storage disease" "0.275815803710352" "0"
## [7,] "lung cancer"          "0.091795796532052" "0.754985574778333"
## [8,] "neoplasm"              "0.0800618415386228" "0.65847824993537"
## [9,] "cervical cancer"       "0.0768036311957259" "0.631680706895131"
## [10,] "lung adenocarcinoma"   "0.074285550791625" "0.610970451599679"
##      RNA Expr. Genetic Assoc.      Somatic Mut. Known Drug Animal Model
## [1,] "0"          "0.547607438737697" "0"          "0"          "0"
## [2,] "0"          "0"                  "0"          "0"          "0"
## [3,] "0"          "0"                  "0"          "0"          "0"
## [4,] "0"          "0"                  "0"          "0"          "0"
## [5,] "0"          "0"                  "0"          "0"          "0"
## [6,] "0"          "0"                  "0"          "0"          "0"
## [7,] "0"          "0"                  "0"          "0"          "0"
## [8,] "0"          "0"                  "0"          "0"          "0"
## [9,] "0"          "0"                  "0"          "0"          "0"
## [10,] "0"         "0"                  "0"          "0"          "0"
##      Affected Pathways
## [1,] "0"
## [2,] "0.45369605355404"
## [3,] "0.45369605355404"
## [4,] "0.45369605355404"
## [5,] "0.45369605355404"
## [6,] "0.45369605355404"
## [7,] "0"
## [8,] "0"
## [9,] "0"
## [10,] "0"

```

El gen **LHX6** está relacionado con varias enfermedades y condiciones, principalmente en el contexto de trastornos neurológicos y neurodegenerativos. A continuación se describen los principales trastornos asociados y sus implicaciones:

- **Auto-reporte de rendimiento educativo:** LHX6 tiene una relación moderada con el rendimiento educativo, lo que podría indicar que este gen tiene alguna influencia en el desarrollo cognitivo y la capacidad de aprendizaje.

- **Enfermedades de almacenamiento lisosomal:** También se ha encontrado una relación con las enfermedades que implican el almacenamiento de sustancias dentro de las células, lo que puede afectar el funcionamiento normal de los órganos.
- **Enfermedades neurodegenerativas:** Trastornos como el Alzheimer y el Parkinson están asociados con LHX6, lo que sugiere que este gen podría estar implicado en la degeneración de las células nerviosas.
- **Esclerosis múltiple:** La esclerosis múltiple, que afecta al sistema nervioso central, también muestra una conexión con LHX6, lo que podría indicar su rol en trastornos autoinmunes que afectan al cerebro.
- **Epilepsia generalizada:** LHX6 tiene una pequeña asociación con formas de epilepsia, como la epilepsia generalizada y la epilepsia con convulsiones febril-plus, lo que podría reflejar su impacto en el funcionamiento eléctrico del cerebro.
- **Neoplasia:** Aunque la asociación es débil, LHX6 también está relacionado con algunos tipos de cáncer, lo que sugiere que podría jugar un papel en el desarrollo de tumores.

Según la información de GeneCards(<https://www.genecards.org/cgi-bin/carddisp.pl?gene=LHX6&keywords=LHX6>) y NCBI, el gen LHX6 produce una proteína que pertenece a una familia de proteínas con un dominio LIM, el cual es rico en cisteínas y tiene la capacidad de unirse al zinc. Esta proteína actúa como un factor de transcripción, lo que significa que regula la expresión de otros genes, y está involucrada en el desarrollo embrionario, especialmente en la formación de la **cabeza**. LHX6 se encuentra principalmente en células mesenquimatosas que provienen de la cresta neural. Además, juega un papel importante en la formación de diferentes tipos de interneuronas corticales y en la migración de precursores de interneuronas GABAérgicas desde una zona llamada el **subpálido hasta la corteza cerebral**.

La información obtenida sobre las enfermedades asociadas con el gen LHX6 se alinea con los datos proporcionados por fuentes confiables como GeneCards y NCBI. Estas fuentes coinciden en que LHX6 está involucrado en diversos trastornos neurológicos, especialmente en la migración y desarrollo de interneuronas en el cerebro.

7.3.3 APLN

diseases\$APLN

\$summary

##	Disease	Overall Score	Literature
##	[1,] "neoplasm"	"0.117569536589689"	"0.966964801352265"
##	[2,] "hepatocellular carcinoma"	"0.109080182332933"	"0.824917958183267"
##	[3,] "type 2 diabetes mellitus"	"0.1078814498911"	"0.887283966488734"


```

## [4,] "coronary artery disease" "0.104630158269966" "0.860543327308197"
## [5,] "atrial fibrillation"      "0.10423918699329" "0.857327736995847"
## [6,] "Obesity"                 "0.104084175941429" "0.856052829946636"
## [7,] "chronic kidney disease"  "0.10276481217925" "0.845201563919629"
## [8,] "Alzheimer disease"       "0.102191009592004" "0.840482255492582"
## [9,] "cancer"                  "0.100962926244952" "0.830381736223972"
## [10,] "Stroke"                 "0.0961588569526662" "0.79087009023433"

##      RNA Expr.      Genetic Assoc. Somatic Mut. Known Drug Animal Model
## [1,] "0"           "0"           "0"           "0"           "0"
## [2,] "0.288900510088732" "0"           "0"           "0"           "0"
## [3,] "0"           "0"           "0"           "0"           "0"
## [4,] "0"           "0"           "0"           "0"           "0"
## [5,] "0"           "0"           "0"           "0"           "0"
## [6,] "0"           "0"           "0"           "0"           "0"
## [7,] "0"           "0"           "0"           "0"           "0"
## [8,] "0"           "0"           "0"           "0"           "0"
## [9,] "0"           "0"           "0"           "0"           "0"
## [10,] "0"          "0"           "0"           "0"           "0"

##      Affected Pathways
## [1,] "0"
## [2,] "0"
## [3,] "0"
## [4,] "0"
## [5,] "0"
## [6,] "0"
## [7,] "0"
## [8,] "0"
## [9,] "0"
## [10,] "0"

```

El **gen APLN** está asociado con diversas enfermedades y tiene un impacto en funciones biológicas importantes. Algunas enfermedades son:

- **Neoplasia (Cáncer):** APLN tiene una relación moderada con el cáncer. Aunque no es una conexión muy fuerte, su papel en procesos biológicos podría influir en el desarrollo de tumores.
- **Diabetes tipo 2:** APLN parece estar involucrado en el control del metabolismo y en la regulación de los niveles de azúcar en la sangre, lo que afecta cómo el cuerpo maneja la insulina.
- **Fibrilación auricular:** Este gen también está asociado con problemas cardíacos, como la fibrilación auricular, que altera el ritmo del corazón. Esto podría estar relacionado con la función de

APLN en la regulación del corazón.

- **Obesidad:** La relación con la obesidad sugiere que el gen podría tener un impacto en el equilibrio energético del cuerpo, influyendo en el almacenamiento y el metabolismo de las grasas.
- **Enfermedades cardiovasculares:** APLN está relacionado con enfermedades como insuficiencia cardíaca, infartos y enfermedades coronarias. Este gen es importante para regular la función del corazón y la formación de vasos sanguíneos.
- **Enfermedades hepáticas y renales:** APLN también está asociado con enfermedades del hígado y los riñones, como el carcinoma hepático y la enfermedad renal crónica, sugiriendo que podría influir en el funcionamiento de estos órganos.

Según la información disponible en GeneCards(<https://www.genecards.org/cgi-bin/carddisp.pl?gene=APLN&keywords=APLN>), NCBI y UniProtKB, este gen está implicado en diversas condiciones de salud y procesos biológicos importantes. A continuación, se detallan las enfermedades relacionadas con APLN y sus funciones:

- **Síndrome Urémico Hemolítico Atípico 1:** APLN se ha relacionado con este síndrome, aunque los mecanismos exactos aún no están completamente definidos.
- **Insuficiencia Placentaria:** El gen también está asociado con problemas en la placenta, lo que puede afectar su desarrollo y función adecuada.

El gen APLN tiene varias funciones importantes en el cuerpo. Ayuda a regular el corazón y la formación de vasos sanguíneos, y también juega un papel en el control de los líquidos del cuerpo, influenciando la sed y la liberación de una hormona llamada vasopresina. Además, APLN participa en la secreción de insulina, lo que afecta cómo se controla el azúcar en la sangre. También tiene un efecto sobre el sistema inmune, especialmente en los bebés, y actúa como un co-receptor en la infección por VIH-1, ayudando a bloquear la entrada del virus en las células.